An extended abstract [RVV24] of this article appears in the proceedings of the 31st ACM Conference on Computer and Communications Security (ACM CCS 2024). This is the full version. The code associated with our paper can be found here.

ISABELLA: Improving Structures of Attribute-Based Encryption Leveraging Linear Algebra

Doreen Riepel[1], Marloes Venema[a,2] and Tanya Verma[3]

[1] University of California San Diego, La Jolla, USA
[2] University of Wuppertal, Wuppertal, Germany
[3] Tinfoil, San Francisco, USA

Abstract. Attribute-based encryption (ABE) is a powerful primitive that has found applications in important real-world settings requiring access control. Compared to traditional public-key encryption, ABE has established itself as a considerably more complex primitive that is additionally less efficient to implement. It is therefore paramount that the we can simplify the design of ABE schemes that are efficient, provide strong security guarantees, minimize the complexity in their descriptions and support all practical features that are desirable for common real-world settings. One of such practical features that is currently still difficult to achieve is multi-authority support. Motivated by NIST's ongoing standardization efforts around multi-authority schemes, we put a specific focus on simplifying the support of multiple authorities in the design of schemes.

To this end, we present ISABELLA, a framework for constructing pairing-based ABE with advanced functionalities under strong security guarantees. At a high level, our approach builds on various works that systematically and generically construct ABE schemes by reducing the effort of proving security to a simpler yet powerful "core" called pair encodings. To support the amount of adaptivity required by multi-authority ABE, we devise a new approach to designing schemes from pair encodings, while still being able to benefit from the advantages that pair encodings provide. As a direct result of our framework, we obtain various improvements for existing (multi-authority) schemes as well as new schemes.

Keywords: attribute-based encryption · multi-authority attribute-based encryption

1 Introduction

Attribute-based encryption (ABE) [SW05] is a powerful primitive that enables more fine-grained enforcement of access control to data. As a cryptographic tool that allows this functionality, ABE has found many applications in practical settings [GPSW06a, ETS18, VAH23, LVV+23], including in cloud settings [KL10, SRGS12, WMZV16] and the Internet of Things [ETS18, VA22]. Most notably, Cloudflare is using Portunus, an access control system based on ABE, on a global scale to terminate TLS connections [LVV+23]. Furthermore, standardization institutes such as NIST [Nat23] and ETSI [ETS18] have recently put efforts in standardizing ABE and its multi-authority variant [Cha07]. Despite the ample research in the modular design of efficient single-authority ABE with strong security guarantees, the design of multi-authority ABE schemes with similarly desirable

E-mail: driepel@ucsd.edu (Doreen Riepel), venema@uni-wuppertal.de (Marloes Venema), tanya@tinfoil.sh (Tanya Verma)
[a]corresponding author

features is still a tedious and difficult task. As a result, few such schemes exist at all, and all of them have limitations, either with respect to security or practicality.

Pair encodings. The most promising frameworks that can be leveraged to simplify the design of pairing-based ABE with advanced functionalities such as multi-authority ABE are based on pair encoding schemes (PES) [Att14, Wee14] with algebraic security notions [AC17b]. Roughly, pair encodings consider "what happens in the exponent" of pairing-based ABE schemes. They allow us to abstract away the complexities of concrete instantations of schemes in pairing groups, and instead focus on the algebraic "core" of the scheme. The most broadly applicable security notion is the symbolic property, first introduced by Agrawal and Chase (AC17) [AC17b] to cover their class of pair encodings, which we refer to as PES-AC17. They show that the symbolic property essentially applies to any scheme that is not algebraically broken. Several frameworks are centered around pair encodings [Att14, Wee14, CGW15, AC16, ABGW17], and more specifically the symbolic property and the AC17 framework [Att19, Amb21, VB24], but the ones that we use as a more direct starting point for our work are

- FABEO [RW22], which focuses on efficiently achieving the strongest notion of security for ABE in the generic group model [Sho97];
- Ven23 [Ven23], which focuses on achieving more advanced functionalities such as multi-authority support;
- ACABELLA [dlPVA23], which focuses on unifying various algebraic security notions in pair encodings and provides an automated tool that helps proving the symbolic property.

The gap. Despite the practical features provided by these frameworks, there remains a gap to be bridged. The most notable shortcoming of FABEO and ACABELLA is that they consider the more restricted class of pair encodings PES-AC17. So far, there exist no practical multi-authority schemes that can be covered by this class of PES-AC17. To address this gap, the Ven23 framework extends the class of encodings, which we refer to as PES-Ven23, to cover many existing multi-authority ABE, as well as several new multi-authority schemes. Additionally, the compiler provided in the Ven23 framework supports the use of full-domain hashes, which is necessary to construct practical (multi-authority) schemes. However, the main drawback of this framework is that the schemes generated with the compiler are proven secure in a weaker security model that captures only static attackers. Moreover, this staticity is also present in the functionality of the compiler itself. Although it is explained how this functional staticity can be addressed in practical applications by making the functionality more adaptive, its security arguments do not extend to the adaptive setting.

The ISABELLA framework. We present ISABELLA, which is a framework based on pair encodings and the symbolic property, which unifies and extends the above three frameworks to simplify the design of ABE schemes with advanced functionalities under strong security guarantees. We achieve this by extending the class PES-Ven23 even further, allowing more efficient instantiations of almost all existing multi-authority ABE schemes via our compiler. Additionally, the ISABELLA compiler does support functional adaptivity by design. Not only does this functional adaptivity allow us to support multi-authority schemes in a fully adaptive setting, but it also allows us to support other types of functional adaptivity, which can be considered an additional feature of our compiler. At the "security core" of our framework, we show that the symbolic property for our class of schemes implies the required security notions to achieve strong security in the generic group model for the compiled schemes. For completeness, we also prove security for our extension of the Ven23

compiler in the weaker model that they consider. By proving security using two different proof methodologies, we obtain security guarantees against multiple different types of attackers, which strengthens the confidence in the schemes' security. To simplify proving the symbolic property, we also extend the ACABELLA tool to cover PES-Ven23 and our class of PES. In this way, we can improve existing ABE schemes as well as design new schemes with advanced functionalities required for practice with high security guarantees.

New results that follow from ISABELLA. Using the ISABELLA compiler, we obtain the following results. For all PES for which the symbolic property was previously proven [AC17b, Att19, Amb21, VA22, VA23, Ven23, VB24], we obtain new security-efficiency trade-offs. More specifically, compared to the Ven23 compiler, we obtain stronger notions of security for all PES-Ven23 schemes, and more efficient instantiations for all multi-authority schemes. Compared to the AC17 compiler [AC17b], we obtain schemes that are at least a factor 2 more efficient in all metrics, at the cost of requiring the generic group model for the security proofs. Furthermore, we show that, for many pair encoding schemes, we can support a functional adaptivity that is valuable for the scalability of schemes in practice. An example of functional adaptivity is that keys can be updated for newly added attributes. Lastly, we introduce two new multi-authority schemes, both of which provide a combination of practical features that has not been achieved before.

1.1 Technical overview of our contributions

We give a technical overview of our contributions. To this end, we first establish some (informal) definitions and notations.

Attribute-based encryption and predicates. Attribute-based encryption is a type of public-key encryption that associates the keys and ciphertexts with predicates. A predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ takes as input a ciphertext predicate $x \in \mathcal{X}$ and a key predicate $y \in \mathcal{Y}$ and evaluates to 1 ("true") if the key predicate satisfies the ciphertext predicate. The idea is that the ciphertext for x can be decrypted by a secret key for y if $P(x,y) = 1$, and that the message remains computationally hidden if not.

Pairing groups. Many ABE schemes use pairings, which is a map $e\colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ over three groups $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ of prime order p, with generators $g \in \mathbb{G}$, $h \in \mathbb{H}$, such that for all $x, y \in \mathbb{Z}_p$, we have $e(g^x, h^y) = e(g,h)^{xy}$.

Starting point: PES-AC17 and PES-Ven23. As a starting point for our framework, we consider PES-AC17. These pair encodings are defined by the master key α, common variables $\mathbf{b} = (b_1, ..., b_n)$, the key encodings $(\mathbf{r}, \mathbf{k}(\alpha, \mathbf{r}, \hat{\mathbf{r}}, \mathbf{b}), y)$ and the ciphertext encodings $(\mathbf{s}, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), x)$, where $\mathbf{b}, \mathbf{r}, \hat{\mathbf{r}}, \mathbf{s}, \hat{\mathbf{s}}$ denote vectors of variables and $\mathbf{k}$ and $\mathbf{c}$ denote vectors of polynomials over the other variables. Compiling the encodings into an ABE scheme results in a scheme that has master public keys, keys and ciphertexts of the following form:

$$\mathsf{MPK} = \left(e(g,h)^{\alpha}, g^{\mathbf{b}}\right), \quad \mathsf{SK}_y = \left(h^{\mathbf{r}}, h^{\mathbf{k}(\mathbf{r},\hat{\mathbf{r}},\alpha,\mathbf{b})}\right),$$
$$\mathsf{CT}_x = \left(M \cdot e(g,h)^{\alpha s}, g^{\mathbf{s}}, g^{\mathbf{c}(\mathbf{s},\hat{\mathbf{s}},\mathbf{b})}\right),$$

where M is the message. Although this results in schemes with a structure that can support some basic functionality, it does not allow us to generate the secret keys by multiple authorities. Roughly, each authority should be able to hold its own copy of a master key α. To enable the support of multiple authorities, the Ven23 compiler extends PES-AC17 to include extra master-key variables $\boldsymbol{\alpha}$ and extra ciphertext encodings $\mathbf{c}'$

instantiated in $\mathbb{G}_T$:

$$\mathsf{MPK} = \left(e(g,h)^{\boldsymbol{\alpha}}, g^{\mathbf{b}}\right), \quad \mathsf{SK}_y = \left(h^{\mathbf{r}}, h^{\mathbf{k}(\mathbf{r},\hat{\mathbf{r}},\boldsymbol{\alpha},\mathbf{b})}\right),$$
$$\mathsf{CT}_x = \left(M \cdot e(g,h)^{c_M}, g^{\mathbf{s}}, g^{\mathbf{c}(\mathbf{s},\hat{\mathbf{s}},\mathbf{b})}, e(g,h)^{\mathbf{c}'(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\alpha})}\right),$$

where c_M denotes a polynomial over $\boldsymbol{\alpha}, \mathbf{s}, \tilde{\mathbf{s}}$. This new structure allows us to construct multi-authority schemes. In fact, many existing multi-authority schemes have a similar structure [LW11, RW15, DKW23a, Ven23]. The drawback is that most multi-authority schemes require a linear number of ciphertext elements in $\mathbb{G}_T$. Because group elements and operations in $\mathbb{G}_T$ are often several factors more expensive than in $\mathbb{G}$ or $\mathbb{H}$ [dlPVA22], this significantly impacts the efficiency of multi-authority schemes compared to their single-authority counterparts.

The ISABELLA class of PES. To address this, we extend the class of PES-Ven23 even further. In particular, we add extra public-key variables called "semi-common" variables $\boldsymbol{\beta}$ and add extra ciphertext components $\mathbf{c}''$ in the source group $\mathbb{G}$.

$$\mathsf{MPK} = \left(e(g,h)^{\boldsymbol{\alpha}}, g^{\mathbf{b}}, g^{\boldsymbol{\beta}}\right), \quad \mathsf{SK}_y = \left(h^{\mathbf{r}}, h^{\mathbf{k}(\mathbf{r},\hat{\mathbf{r}},\boldsymbol{\alpha},\boldsymbol{\beta},\mathbf{b})}\right),$$
$$\mathsf{CT}_x = \left(M \cdot e(g,h)^{c_M}, g^{\mathbf{s}}, g^{\mathbf{c}(\mathbf{s},\hat{\mathbf{s}},\mathbf{b})}, e(g,h)^{\mathbf{c}'(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\alpha},\boldsymbol{\beta})}, g^{\mathbf{c}''(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\beta})}\right).$$

To improve the efficiency of the aforementioned schemes, the idea is roughly that the ciphertext components generated via $\mathbf{c}'$ are moved to $\mathbf{c}''$. To enable this, we require new proof techniques, which we will explain in more detail later, in Sections 3 and 5.

Adding more (advanced) functionalities. Equipped with a "core compiler" for ABE, we can extend it with more advanced functionalities. Like the Ven23 compiler, we also allow that certain "variables" are generated implicitly via full-domain hashes, and we also facilitate a more flexible instantiation of the pair encodings in the groups. For example, we may want to put $g^{\mathbf{s}}$ in $\mathbb{H}$ and $h^{\mathbf{k}(\mathbf{r},\hat{\mathbf{r}},\boldsymbol{\alpha},\boldsymbol{\beta},\mathbf{b})}$ in $\mathbb{G}$ to speed up decryption [dlPVA22]. Furthermore, hashes allow the support of multiple advanced features, such as unlimited use of attributes (in the sense that any string can be used as attribute). Hashes are also an important key ingredient to the stronger notion of decentralized multi-authority ABE [LW11], which allows authorities to act more independently and autonomously than in schemes that require interaction among the authorities [Cha07, CC09, MJ18]. To facilitate this, we define mappings that specify for all encoding variables and polynomials in which group they are instantiated, and for the variables in particular, whether they are generated via a full-domain hash.

Adaptively generating keys. To support multiple authorities, we need to be able to split the public keys and secret keys, so that each authority can manage its own part of the MPK (and associated MSK) and generate secret keys for its own "part of the key predicate y". Although Ven23 does give concrete ideas on how to model this from a functional point of view for a specific class of multi-authority ABE, the framework does not describe how the encodings interact with ABE syntax for such advanced functionalities. Furthermore, the secret keys are split in partial secret keys *after* being compiled. From a technical point of view, this does not allow us to prove adaptive security, because the attacker cannot adaptively request partial secret keys from different authorities. Hence, we devise a new approach that allows us to prove adaptive security and gives us a concrete (yet generic) syntax that interacts naturally with existing syntax such as that of multi-authority ABE. Our technique is also more generically applicable to any predicate P, whereas the Ven23 framework only considers multi-authority ciphertext-policy ABE.

Split-predicate mold. Our new approach considers the use of what we call a "split-predicate mold", which we can place on top of the PES to show that it satisfies a certain structure. Roughly, the split-predicate mold consists of three components: a description of how the *inputs* to the public-key and (secret-)key encodings are split in parts, a description of how the *outputs*, i.e., the encodings, are split in parts, and how the split inputs interact with the split outputs. In addition, we require an auxiliary input for the key generation, which connects the keys together, so that they cannot be combined with "other keys" (which would break the collusion resistance property). To enable that keys are strongly connected, the auxiliary input can be used to generate implicitly key variables from a full-domain hash. In fact, we require that the only key variables $\mathbf{r}$ that are shared among different splits of the original key must be generated via a hash in the compiled scheme. Further, the split keys need to be completely disjoint and, when "aggregated" for the original predicate, they should be identical, i.e., symbolically indistinguishable, from the original encodings (for correctness and security).

Splitting the key predicates in sub-predicates. The inputs that need to be split include key predicates y and their associated spaces $\mathcal{Y}$, which can be sets, Boolean formulas or something else, and thus, do not have a fixed structure that we can exploit. Therefore, we formalize the notion of "sub-predicate spaces" and "aggregation function", which roughly describe the properties that the sub-predicate spaces need to have to be suitable for our mold and compiler. A description of the sub-predicate spaces and aggregation functions need to be given as part of the mold. As part of our framework, we give several examples of commonly used predicates and sub-predicate spaces that can be used to achieve functionalities such as multi-authority ABE, but also other types of functionalities.

The ISABELLA compiler. Armed with a definition for PES, mappings that specify how the variables are instantiated in the pairing groups and a split-predicate mold that describes how the predicates and encodings are split, we can now define our compiler in such a way that the syntax matches the split-predicate mold. Roughly, the setup and key generation algorithms are split in an analogous way as the encodings, as well as their inputs. Hence, if the predicate for the original PES is a "multi-authority predicate", i.e., specifies how authorities are related to the predicate, and the key generation algorithms take as input a sub-predicate that is a valid sub-predicate for the multi-authority predicate, then we obtain a multi-authority scheme. Since the naming conventions of our algorithms are generic and therefore do not match any existing ABE syntax, we further illustrate how our syntax corresponds to existing syntax. We give a generic description of how multi-authority ABE can be constructed using the ISABELLA compiler, and specifically, how it interacts with multi-authority ABE syntax. We also show how it could interact with other types of functionality.

Functional adaptivity. An advantage of our generic formalizations is that they can also be applied for other purposes than the support of multiple authorities. For example, it can be used to generate keys of single-authority schemes in which y is an attribute set in an "attribute-wise" fashion rather than in one query. We call such functionalities "functional adaptivity", as it enables users to request keys more adaptively than schemes typically allow (both from a functional and security point of view). Such features are beneficial in practice, because it allows users to update their keys rather than requesting entirely fresh keys for each update of the key predicate. To the best of our knowledge, the only examples of ABE schemes that could already support such a feature are multi-authority ABE schemes, and in almost all cases except one (that we know of) [DKW23b], the security proofs do not model this adaptivity.

Proving security. To generically prove security for schemes generated with the ISABELLA compiler, we use the special selective symbolic property (SSSP). SSSP is an adaptation of the symbolic properties in PES-AC17 and PES-Ven23 adapted to our class of PES. From this property, we obtain two security results with our compiler: fully adaptive security in the generic group model (GGM) [Sho97] and static security from q-type assumptions. In the fully adaptive model, the attacker gets to make queries for the key and ciphertext predicates after observing public keys, secret keys and the challenge ciphertext. Static attackers, on the other hand, need to commit to the key or ciphertext predicates (or both) before even seeing any public keys. (Similarly, we distinguish between static and adaptive corruptions of public keys, which is a security feature that multi-authority ABE provides.)

Security in the fully adaptive setting is generally harder to achieve, and therefore, to allow for the best efficiency, we restrict to adversaries in the GGM that only exploit the group structure. We then also prove static security in the standard (or random oracle) model[1] under q-type assumptions. This provides additional confidence: any attacker that might be able to attack the schemes (because we do not model security against it) would need to be very sophisticated. On the one hand, it needs to be fully adaptive (exploiting that it can observe keys and ciphertexts before formulating that it wants to attack a certain ciphertext predicate). On the other hand, it should also exploit the group structures that are used when instantiating the scheme rather than the group and pairing operations. (Of course, attacks on the *implementations* of schemes provided by our compiler might still be possible.)

Our proof strategy. Our proof strategy follows closely the structural layers of our compiler. At the core, we have PES and SSSP, which consider security for the "stripped down" version of the scheme (that in particular does not consider advanced functionalities). This allows us to leverage existing results in the field of PES and the symbolic property, and extend the results to apply to our larger class of schemes. To achieve this, we strengthen the relationship between the symbolic property and the algebraic notions of security that are used in GGM proofs. We use this "security core" in our proofs by reducing the schemes, provided by our compiler and the split-predicate mold, to itself, but as if no mold was applied at all (or alternatively, a trivial mold that splits everything in one part). Intuitively, this part of the reduction uses that the split keys (generated using split inputs) are indistinguishable from keys that were generated directly via the PES (instantiated with a trivial mold and using the original predicate). Furthermore, full-domain hashes (if present) are modeled as random oracles which allows us to program the components that were generated by it in the same way we would have if they were not generated via a hash. To enable flexible instantiation of the PES in the groups, we use techniques based on the security models in which we prove security.

Leveraging linear algebra. Our "security core", consisting of PES and SSSP, is largely powered by linear algebra. At a high level, SSSP states that there must exist some vectors and matrices (with some additional properties) that, when substituted into the polynomials in the pair encodings, evaluate to all-zero vectors. (The only exception is c_M, which should be nonzero.) This property is applied in its literal sense in the security reduction (to a variant of decisional bilinear Diffie-Hellman). Roughly, the substitution of vectors and matrices corresponds to the programming of the components associated with the variables (additionally using terms of the security assumption). The symbolic property ensures that components associated with the key and ciphertext polynomials can be programmed by canceling out the components that cannot be programmed using the assumption.

[1] The random oracle model (which is a strictly milder heuristic than the generic group model [ZZ23]) is required only if the scheme uses a full-domain hash function.

For the GGM proofs, we require another algebraic security notion. In particular, this algebraic notion states that c_M should not be recoverable from key and ciphertext encodings for y and x for which the predicate is false by computing linear combinations over products of those encodings. In fact, this property should hold, even when considering multiple key predicates $y_1, ..., y_{n_k}$ (to model security against colluding users). To prove multi-key security, FABEO gives a stronger notion of security for the single-key setting that implies security in the multi-key setting. ACABELLA later shows that the multi-key security property is implied by a slightly stronger version of the symbolic property for PES-AC17 (which considers security in the single-key setting). It is also shown that the variant of the symbolic property is significantly weaker than the FABEO security property. In fact, a similar such property adapted to our class of schemes would be too strong to prove it for the schemes considered in this paper or in Ven23. Hence, we extend the ACABELLA results to the class of PES-Ven23 and ours and prove that our version of SSSP implies multi-key (and multi-ciphertext) security.

ACABELLA extension. In addition to SSSP being broadly applicable, another advantage is that its algebraic character can be exploited in the design of automated tools. ACABELLA [dlPVA23] provides a tool that generates, on input a description of a pair encoding scheme, a "proof" for SSSP. A proof here consists of the aforementioned substitution vectors and matrices. We extend the tool to cover PES-Ven23 and our class of PES. To enable this, we again leverage linear algebra. Because some of the requirements for our definition of SSSP are more abstract than that of PES-AC17, we provide new techniques to compute the proofs. The code of the ACABELLA extension is available at https://github.com/lincolncryptools/ISABELLA.

Improving existing schemes and designing new schemes. As a direct result of our framework, we improve (on) existing schemes in the efficiency and security. We also include two new pair encoding schemes that can be instantiated with our compiler that improve on the state of the art in the field of multi-authority ABE.

1.2 Roadmap

We start by providing necessary preliminaries in Section 2, where we define (multi-authority) attribute-based encryption and predicates. In Section 3 we give our class of pair encoding schemes PES-ISA and several notions of security: the special symbolic property (Section 3.1), multi-key and multi-ciphertext security (Section 3.2) and security in the matrix notation (Section 3.3). We also extend the two latter ones to capture corruptions (Section 3.4) and show they are equivalent to the special symbolic property (Section 3.5). Each of these notions has their advantage in security proofs. In particular, our computer-assisted proofs use the matrix notation, which we explain in Section 3.6, with a more detailed description and additional proofs in Appendices A and B.

In Section 4 we give our compiler. When instantiating a PES-ISA in a pairing group, several choices have to be made to ensure correctness and security. Therefore, we first discuss distributions of encodings over the pairing group and how to use full-domain hashes (Section 4.1). Further, since PES-ISA is static, we define a split-predicate mold (Section 4.2) to capture more advanced schemes that support functional adaptivity. Putting these together, we get the final compiler (Section 4.3). Section 5 is dedicated to the security of the compiler. We first provide a detailed description of the security model (Section 5.1), capturing different levels of adaptivity. Then we give the theorem statements for static security under q-type assumptions (Section 5.2) and adaptive security in the generic group model (Section 5.3), with full proofs in Appendices C and D.

In Section 6 we show how to use our compiler to construct multi-authority ABE (Section 6.1) and schemes with functional adaptivity (Section 6.2), e.g., those supporting attribute-wise or label-wise key generation and negations. In addition to that, we elaborate in Appendix E how exisiting pair encoding schemes can be instantiated using our compiler and we give new pair encoding schemes and instantiations in Appendix F. Finally, we give a concrete efficiency comparison for decentralized ciphertext-policy ABE schemes in Appendix G.

2 Preliminaries

2.1 Notation

We use λ to denote the security parameter. A negligible function parametrized by λ is denoted as $\mathrm{negl}(\lambda)$. If an element x is chosen uniformly at random from a finite set S, then we denote this as $x \in_R S$. If an element x is produced by running algorithm Alg, then we denote this as $x \leftarrow \mathrm{Alg}$. We use $\mathbb{Z}_p = \{x \in \mathbb{Z} \mid 0 \leq x < p\}$ for the set of integers modulo p. For integers $a < b$, we denote $[a, b] = \{a, a+1, ..., b-1, b\}$, $[b] = [1, b]$ and $\overline{[b]} = [0, b]$. We use boldfaced variables $\mathbf{A}$ and $\mathbf{v}$ for matrices and vectors, respectively, where $(\mathbf{A})_{i,j}$ denotes the entry of $\mathbf{A}$ in the i-th row and j-th column, and $(\mathbf{v})_i$ denotes the i-th entry of $\mathbf{v}$. We use $\mathbf{A}_i$ to denote the i-th row of matrix $\mathbf{A}$. We denote $a \ : \ \mathbf{A}$ to substitute variable a by a matrix or vector $\mathbf{A}$. We define $\mathbf{1}_{i,j}^{d_1 \times d_2} \in \mathbb{Z}_p^{d_1 \times d_2}$ as the matrix with 1 in the i-th row and j-th column, and 0 everywhere else, and similarly $\mathbf{1}_i^{d_1}$ and $\overline{\mathbf{1}}_i^{d_2}$ as the row and column vectors with 1 in the i-th entry and 0 everywhere else. If some algorithm yields no output or outputs an error message, then we use $\perp$ to indicate this. We use $a \| b$ to indicate that two strings a and b are concatenated. We use $\mathrm{span}(\mathbf{v}) = \{\sum_i c_i(\mathbf{v})_i \mid \forall i[c_i \in \mathbb{Z}_p]\}$ and $\mathrm{span}(\mathbf{A}) = \{\sum_i c_i \mathbf{A}_i \mid \forall i[c_i \in \mathbb{Z}_p]\}$ to denote the span of vector $\mathbf{v}$ and matrix $\mathbf{A}$, respectively. We use $\mathrm{Ker}(\mathbf{A}) = \{\mathbf{w} \mid \mathbf{A} \cdot \mathbf{w}^\intercal = \mathbf{0}\}$ to denote the kernel of matrix $\mathbf{A}$. If we want to construct a vector from elements in some set $\mathcal{S}$, then we denote this as $(\mathcal{S})$, e.g., $(\{x_1, ..., x_n\})$. A special variant of this is a truncated vector, i.e., let $\mathbf{v}$ denote a vector and $\mathcal{I}$ is a set of indices of $\mathbf{v}$, then $\mathbf{v}_\mathcal{I} = (\{(\mathbf{v})_i \mid i \in \mathcal{I}\})$ denotes the vector truncated to the indices in $\mathcal{I}$. Similarly, we define $x_\mathcal{I} = \{x_i \mid i \in \mathcal{I}\}$.

2.2 Access structures

We represent access policies $\mathbb{A}$ by linear secret sharing scheme (LSSS) matrices, which support monotone span programs [Bei96].

Definition 1 (Access structures represented by LSSS [GPSW06b])**.** An access structure is represented as a pair $\mathbb{A} = (\mathbf{A}, \rho)$ such that $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ is an LSSS matrix, where $n_1, n_2 \in \mathbb{N}$, and ρ is a function that maps its rows to attributes in the universe. Then, for some vector $\mathbf{v} = (s, v_2, ..., v_{n_2}) \in_R \mathbb{Z}_p^{n_2}$, the i-th share of secret s generated by this matrix is $\lambda_i = \mathbf{A}_i \mathbf{v}^\intercal$, where $\mathbf{A}_i$ denotes the i-th row of $\mathbf{A}$. In particular, if $\mathcal{S}$ satisfies $\mathbb{A}$, then there exist a set of rows $\Upsilon = \{i \in [n_1] \mid \rho(i) \in \mathcal{S}\}$ and coefficients $\varepsilon_i \in \mathbb{Z}_p$ for all $i \in \Upsilon$ such that $\sum_{i \in \Upsilon} \varepsilon_i \mathbf{A}_i = (1, 0, ..., 0)$, and by extension $\sum_{i \in \Upsilon} \varepsilon_i \lambda_i = s$, holds. If $\mathcal{S}$ does not satisfy $\mathbb{A}$, there exists $\mathbf{w} = (1, w_2, ..., w_{n_2}) \in \mathbb{Z}_p^{n_2}$ such that $\mathbf{A}_i \mathbf{w}^\intercal = 0$ for all $i \in \Upsilon$ [Bei96].

2.3 Pairings (or bilinear maps)

We define a pairing to be an efficiently computable map e on three groups $\mathbb{G}, \mathbb{H}$ and $\mathbb{G}_T$ of prime order p, so that $e \colon \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$, with generators $g \in \mathbb{G}, h \in \mathbb{H}$ is such that (i) for all $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, h^b) = e(g, h)^{ab}$ (bilinearity), and (ii) for $g^a \neq 1_\mathbb{G}, h^b \neq 1_\mathbb{H}$, it holds that $e(g^a, h^b) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}'}$ denotes the unique identity element of the associated group $\mathbb{G}'$ (non-degeneracy). We denote by $\mathcal{PG} = (\mathbb{G}, \mathbb{H}, \mathbb{G}_T, p, e, g, h)$ the description of

a pairing group and by PGGen a PPT algorithm that on input the security parameter λ, returns a pairing group $\mathcal{PG}$. Furthermore, we use the implicit representation used for group elements [EHK+13]. In particular, we use $[x]_{\mathbb{G}'}$ to denote the element $(g')^x$, where $g' \in \mathbb{G}'$ is the generator of some group $\mathbb{G}' \in \{\mathbb{G}, \mathbb{H}, \mathbb{G}_T\}$.

2.4 Attribute-based encryption

We define ABE for predicates following [Att14, AC17b].

Predicate family. A *predicate* $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ is a map over the ciphertext attribute space $\mathcal{X}$ and the key attribute space $\mathcal{Y}$ such that, for $x \in \mathcal{X}, y \in \mathcal{Y}$, $P(x,y) = 1$ if and only if the predicate evaluates to true for x and y.

Syntax. An attribute-based encryption scheme ABE for a predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ over a message space $\mathcal{M}$ consists of four algorithms:

- $\mathsf{Setup}(\lambda) \to (\mathsf{MPK}, \mathsf{MSK})$: On input the security parameter λ, this probabilistic algorithm generates the master public key MPK and the master secret key MSK.
- $\mathsf{KeyGen}(\mathsf{MSK}, y) \to \mathsf{SK}_y$: On input the master secret key MSK and some $y \in \mathcal{Y}$, this probabilistic algorithm generates a secret key SK_y.
- $\mathsf{Encrypt}(\mathsf{MPK}, x) \to (\mathsf{CT}_x, K)$: On input the master public key MPK and some $x \in \mathcal{X}$, this probabilistic algorithm generates a ciphertext CT_x and an encapsulated key K.
- $\mathsf{Decrypt}(\mathsf{MPK}, \mathsf{SK}_y, \mathsf{CT}_x) \to K$: On input the master public key MPK, the secret key SK_y, and the ciphertext CT_x, if $P(x,y) = 1$, then it returns K. Otherwise, it returns $\perp$.

Note that we are using a KEM-style definition from which one can generically construct an encryption scheme.

Correctness. For all $M \in \mathcal{M}$, $x \in \mathcal{X}$, and $y \in \mathcal{Y}$ with $P(x,y) = 1$,

$$\Pr[(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(\lambda); (\mathsf{CT}_x, K) \leftarrow \mathsf{Encrypt}(\mathsf{MPK}, x) : \mathsf{Decrypt}(\mathsf{MPK}, \mathsf{KeyGen}(\mathsf{MSK}, y), \mathsf{CT}_x) \neq K] \leq \mathrm{negl}(\lambda).$$

Security. We define the security game capturing security against chosen-plaintext attacks (CPA) as in [AC17b]. The game $\mathsf{IND\text{-}CPA}$ between a challenger and an adversary A proceeds as follows:

- **Setup phase:** The challenger runs $\mathsf{Setup}(\lambda)$ to obtain MPK and MSK, and sends the master public key MPK to A.
- **First query phase:** A queries secret keys for $y \in \mathcal{Y}$, and obtains $\mathsf{SK}_y \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, y)$ in response.
- **Challenge phase:** A specifies some $x^* \in \mathcal{X}$ such that for all y in the first key query phase, we have $P(x^*, y) = 0$, and sends y to the challenger. The challenger flips a coin, i.e., $\delta \in_R \{0,1\}$, computes $(\mathsf{CT}_{x^*}, K_0) \leftarrow \mathsf{Encrypt}(\mathsf{MPK}, x^*)$ and $K_1 \in_R \mathcal{K}$, and sends the ciphertext $(\mathsf{CT}_{x^*}, K_\delta$ to A.
- **Second query phase:** This phase is identical to the first query phase, with the additional restriction that the adversary can only query $y \in \mathcal{Y}$ such that $P(x^*, y) = 0$.
- **Decision phase:** A outputs a guess δ' for δ.

Definition 2 (Security against CPA)**.** Let ABE be an ABE scheme and let A be an adversary in the above game. We define the advantage of A as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ABE},A}(\lambda) \coloneqq \left| \Pr[\delta' = \delta] - \frac{1}{2} \right| .$$

We say that ABE is fully secure if all polynomial-time adversaries A have at most a negligible advantage in this security game, i.e., $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ABE},A}(\lambda) \leq \mathrm{negl}(\lambda)$.

Similarly, we can define weaker variants of the above definition. In the *selective* security model, the adversary commits to the predicate $x^* \in \mathcal{X}$ before the Setup phase. In the *co-selective* security model, the adversary commits to all $y \in \mathcal{Y}$ before the Setup phase. In the *static* security model, the attacker commits to $x^* \in \mathcal{X}$ and all $y \in \mathcal{Y}$ before the Setup phase.

2.5 Specific predicate types

Ciphertext-policy and key-policy ABE. A specific instance of ABE is ciphertext-policy ABE, where the key predicate y is a set of attributes $\mathcal{S}$ over attribute universe $\mathcal{U}$, and the ciphertext predicate x is an access policy $\mathbb{A} = (\mathbf{A}, \rho)$, in this work represented as LSSS matrices (Definition 1), which is known to support all monotone Boolean formulas. We denote $\mathcal{X}_{\text{CP-basic}}$ as the basic ciphertext predicate space containing all the tuples $(\mathbf{A}, \rho)$ and $\mathcal{Y}_{\text{CP-basic}}$ as the basic key predicate space containing all $\mathcal{S} \subseteq \mathcal{U}$. The dual variant of CP-ABE, i.e., key-policy ABE (KP-ABE), sets $\mathcal{X}_{\text{KP-basic}} \coloneqq \mathcal{Y}_{\text{CP-basic}}$ and $\mathcal{Y}_{\text{KP-basic}} \coloneqq \mathcal{X}_{\text{CP-basic}}$.

Supporting negations. The type of negations that we support is the same as the OSWOT-type, which was first proposed by Attrapadung and Tomida [AT20]. This type of negation is enforced on the value within an attribute type. In particular, a negation is satisfied by a set $\mathcal{S}$ if $\mathcal{S}$ contains at least one attribute with the same label as the negated attribute and none of the values of the attributes with the same label is equal to the negated attribute. For this predicate, we require the ciphertext space $\mathcal{X}_{\text{CP-basic}}$ to be extended with the map ρ_{lab} (that maps the rows of the policy matrix to labels), and the map ρ' (that specifies for each row whether the attribute is negated or not), i.e., $\mathcal{X}_{\text{CP-not}} = \{(\mathbf{A}, \rho, \rho', \rho_{\mathsf{lab}}) \mid (\mathbf{A}, \rho) \in \mathcal{X}_{\text{CP-basic}}, \rho' \colon [n_1] \to \{0,1\}, \rho_{\mathsf{lab}} \colon [n_1] \to \mathcal{L}\}$. We extend the sets in $\mathcal{Y}_{\text{CP-basic}}$ with an additional entry in the tuple, i.e., $\mathcal{Y}_{\text{CP-not}} = \{\mathcal{S} \mid \mathcal{S} \subseteq \mathcal{L} \times \mathcal{U}\}$. Hence, each attribute in the set is now represented as a pair $(\mathsf{lab}, \mathsf{att})$ with label $\mathsf{lab} \in \mathcal{L}$ and attribute value $\mathsf{att} \in \mathcal{U}$. We define the predicate on these ciphertext and key predicate spaces as $P_{\text{CP-not}} \colon \mathcal{X}_{\text{CP-not}} \times \mathcal{Y}_{\text{CP-not}} \to \{0,1\}$, where $P_{\text{CP-not}}(x, y) = 1$ if and only if, for $\Upsilon = \{j \in [n_1] \mid (\rho_{\mathsf{lab}}(j), \rho(j)) \in \mathcal{S} \wedge \rho'(j) = 1\}$ and $\overline{\Upsilon} = \{j \in [n_1] \mid (\rho_{\mathsf{lab}}(j), \rho(j)) \notin \mathcal{S} \wedge \rho'(j) = 0 \wedge \exists \mathsf{att} : (\rho_{\mathsf{lab}}(j), \mathsf{att}) \in \mathcal{S}\}$, there exist $\{\varepsilon_j\}_{j \in \Upsilon \cup \overline{\Upsilon}}$ with $\sum_{j \in \Upsilon \cup \overline{\Upsilon}} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$.

2.6 Multi-authority ABE

In the multi-authority setting, the Setup is split in two algorithms, where one is run globally and the other one is run by each authority in the system. Furthermore, each authority executes the key generation with its own input.

Syntax. A multi-authority ABE scheme $\mathsf{MA\text{-}ABE}$ for a predicate $P \colon \mathcal{X}_{\text{MA}} \times \mathcal{Y}_{\text{MA}} \to \{0,1\}$ and for authorities $\mathcal{A}_1, \ldots, \mathcal{A}_{n_{\text{aut}}}$ such that each authority $\mathcal{A}_i$ manages a sub-predicate space $\mathcal{Y}_{\text{SA}}$ of $\mathcal{Y}_{\text{MA}}$, consists of five algorithms:

- $\mathsf{GlobalSetup}(\lambda) \to \mathrm{GP}$: On input the security parameter λ, this algorithm generates the global domain parameters GP.
- $\mathsf{AuthoritySetup}(\mathrm{GP}) \to (\mathcal{A}, \mathsf{MPK}_{\mathcal{A}}, \mathsf{MSK}_{\mathcal{A}})$: This algorithm takes as input the global parameters and outputs an authority identifier $\mathcal{A}$ and its master public and secret key.
- $\mathsf{KeyGen}(\mathcal{A}, \mathsf{MSK}_{\mathcal{A}}, \mathsf{GID}, y_{\mathsf{GID},\mathcal{A}}) \to \mathsf{SK}_{\mathsf{GID},\mathcal{A},y_{\mathsf{GID},\mathcal{A}}}$: This algorithm takes as input an authority identifier, the authority's master secret key, a user identifier GID and some $y_{\mathsf{GID},\mathcal{A}} \in \mathcal{Y}$. It outputs a secret key $\mathsf{SK}_{\mathsf{GID},\mathcal{A},y_{\mathsf{GID},\mathcal{A}}}$ for this user.
- $\mathsf{Encrypt}(\{\mathcal{A}_i, \mathsf{MPK}_{\mathcal{A}_i}\}_i, x) \to (\mathsf{CT}_x, K)$: This algorithm takes as input a set of authority identifiers, their master public keys and some $x \in \mathcal{X}$. It outputs a ciphertext CT_x and an encapsulated key K.
- $\mathsf{Decrypt}(\mathrm{GP}, \{\mathsf{SK}_{\mathsf{GID},\mathcal{A}_j,y_{\mathsf{GID},\mathcal{A}_j}}\}_{j\in\mathcal{J}'}, \mathsf{CT}_x) \to K$: This algorithm takes as input a set of user secret keys issued by authorities $\mathcal{J}' \subseteq [n_{\mathrm{aut}}]$ and a ciphertext. If decryption is successful, then it outputs an encapsulated key K.

Sometimes interaction between authorities is required and the number of authorities may be fixed in advance.

Extending predicates with multi-authority support. Let $\mathcal{AID} = \{0,1\}^*$ be the set of authority identifiers. We extend the predicate spaces for single-authority schemes as follows. For the policies, we include the $\tilde{\rho}\colon [n_1] \to \mathcal{AID}$, which maps the rows of the policy to authority identifiers. For the sets, we extend the attribute representation with an additional entry for the authority, e.g., $(l, \mathsf{att}) \in \mathcal{S}$ denotes an attribute att in $\mathcal{U}$ managed by authority $\mathcal{A}_l \in \mathcal{AID}$. We prefix the indices of the extended predicate spaces with "MA-", e.g., $\mathcal{X}_{\text{MA-CP-basic}} = \{(\mathbf{A}, \rho, \tilde{\rho}) \mid (\mathbf{A}, \rho) \in \mathcal{X}_{\text{CP-basic}}, \tilde{\rho}\colon [n_1] \to \mathcal{AID}\}$ and $\mathcal{Y}_{\text{MA-CP-basic}} = \{\mathcal{S} \mid \mathcal{S} \subseteq \mathcal{AID} \times \mathcal{U}\}$. Subsequently, we define the predicate $P_{\text{MA-CP-basic}}\colon \mathcal{X}_{\text{MA-CP-basic}} \times \mathcal{Y}_{\text{MA-CP-basic}} \to \{0,1\}$ to be $P_{\text{MA-CP-basic}}((\mathbf{A}, \rho, \tilde{\rho}), \mathcal{S}) = 1$ if and only if there exists $\Upsilon = \{j \in [n_1] \mid (\tilde{\rho}(j), \rho(j)) \in \mathcal{S}\}$ and $\{\varepsilon_j\}_{j\in\Upsilon}$ such that $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$. Note that our notation implies that all authorities in the system share the same attribute universe. This is only syntactic: any requirements on the universes managed by the authorities (e.g., the universes must be disjoint) can be made by restricting the space $\mathcal{Y}_{\text{MA-CP-basic}}$. However, note that the schemes provided in this paper do not require this restriction.

Multi-authority key-policy ABE. We further extend the predicate spaces for KP-ABE to the multi-authority setting by constructing an AND-statement over policies for each authority. We denote this as $\mathcal{Y}_{\text{MA-KP-basic}} = \{\{y_l\}_{l\in\mathcal{S}'_{\mathcal{A}}} \mid \mathcal{S}'_{\mathcal{A}} = \{\mathcal{A}_l \in \{0,1\}^*\}_{l\in[n_{\mathrm{aut}}]}, y_l \in \mathcal{Y}_{\text{KP-basic}}\}$ and $\mathcal{X}_{\text{MA-KP-basic}} = \{\{x_l\}_{l\in\mathcal{S}_{\mathcal{A}}} \mid \mathcal{S}_{\mathcal{A}} = \{\mathcal{A}_l \in \{0,1\}^*\}_{l\in[n_{\mathrm{aut}}]}, y_l \in \mathcal{Y}_{\text{KP-basic}}\}$, and $P_{\text{MA-KP-basic}}\colon \mathcal{X}_{\text{MA-KP-basic}} \times \mathcal{Y}_{\text{MA-KP-basic}} \to \{0,1\}$ with $P_{\text{MA-KP-basic}}(x, y) = 1$ with $x = \{x_l\}_{l\in\mathcal{S}_{\mathcal{A}}}$ and $y = \{y_l\}_{l\in\mathcal{S}'_{\mathcal{A}}}$, iff $\mathcal{S}_{\mathcal{A}} \subseteq \mathcal{S}'_{\mathcal{A}}$ and $P_{\text{KP-basic}}(x_l, y_l) = 1$ for all $l \in \mathcal{S}_{\mathcal{A}}$. Intuitively, the key space describes a set of authority-specified policies, and the ciphertext space describes for which subset of authorities the keys must satisfy the policies.

Security. The security model extends the one for single-authority ABE by additional oracles to capture the ability to create (and corrupt) authorities. It is in fact very similar to our general security game in Section 5.1. Similar to standard ABE, the adversary may have to provide key or ciphertext predicates in advance. In the multi-authority setting, the adversary may also commit to the set of corrupted authorities. There exist several naming conventions for this in the literature (cf. e.g., [RW15, AG23]).

Corruption and its effect on the predicate. Multi-authority ABE typically provides security against corruption, i.e., if the corrupted authorities and the decryption capabilities of the generated keys do not yield sufficient knowledge to satisfy the predicate, then the scheme should still be secure. For example, if we consider the policy $\mathsf{att}_1 \wedge \mathsf{att}_2 \wedge \mathsf{att}_3$, where att_1 is mapped to authority $\mathcal{A}_1$, att_2 to $\mathcal{A}_2$ and att_3 to $\mathcal{A}_3$, authority $\mathcal{A}_1$ is corrupted and the adversary has a key for att_2, then the message should remain hidden.

3 Our class of pair encoding schemes

Our compiler extends the class of pair encoding schemes covered by the Ven23 compiler to a broader class of PES which we call PES-ISA. Roughly, the definition of PES is adapted in such a way that the associated the master public key, the secret keys and the ciphertexts have the following form:

$$\mathsf{MPK} = \big(e(g,h)^{\boldsymbol{\alpha}}, (g')^{\mathbf{b}}, (g')^{\boldsymbol{\beta}}\big), \qquad \mathsf{SK}_y = \big((g')^{\mathbf{r}}, (g')^{\mathbf{k}(\mathbf{r},\hat{\mathbf{r}},\boldsymbol{\alpha},\boldsymbol{\beta},\mathbf{b})}\big),$$
$$\mathsf{CT}_x = \big(M \cdot e(g,h)^{c_M}, (g')^{\mathbf{s}}, (g')^{\mathbf{c}(\mathbf{s},\hat{\mathbf{s}},\mathbf{b})}, e(g,h)^{\mathbf{c}'(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\alpha},\boldsymbol{\beta})}, (g')^{\mathbf{c}''(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\beta})}\big),$$

(where g' indicates that either $g' = g$ or $g' = h$ for each entry of the vector in the exponent). The associated pair encoding are denoted "in the exponent", and are supposed to illustrate the main differences between PES-Ven23 and PES-ISA. Compared to the Ven23 compiler, ours includes variables $\boldsymbol{\beta}$. In contrast to the variables $\boldsymbol{\alpha}$, these can also occur in the ciphertext encodings that are instantiated in the source group, whereas the $\boldsymbol{\alpha}$ variables occur exclusively in the ciphertext encodings that are instantiated in the target group. More formally, like the $\boldsymbol{\alpha}$ variables, these $\boldsymbol{\beta}$ variables occur as lone variables in the keys and as non-lone variables in the ciphertexts. Hence, structurally speaking, the $\boldsymbol{\beta}$ encodings are closer to $\boldsymbol{\alpha}$ than to $\mathbf{b}$, and the difference between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is mostly syntactical. More concretely, PES-ISA is defined as follows.

Definition 3 (Our class of pair encoding schemes (PES-ISA))**.** A pair encoding scheme $\Gamma_{\text{PES-ISA}}$ for the class of schemes covered by our compiler, and for a predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and a prime integer $p \in \mathbb{N}$, is given by four deterministic polynomial-time algorithms as described below.

- $\text{Param}(\text{par}) \to (n_\alpha, n_b, n_\beta, \boldsymbol{\alpha}, \mathbf{b}, \boldsymbol{\beta})$: On input parameters par, the algorithm outputs $n_\alpha, n_b, n_\beta \in \mathbb{N}$ that specify the number of master key variables, common variables and semi-common variables, respectively, which are denoted as $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_{n_\alpha})$, $\mathbf{b} = (b_1, ..., b_{n_b})$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_{n_\beta})$, respectively.
- $\text{EncKey}(y) \to (m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b}))$: On input $y \in \mathcal{Y}$, this algorithm outputs a vector of polynomials $\mathbf{k} = (k_1, ..., k_{m_3})$ defined over non-lone variables $\mathbf{r} = (r_1, ..., r_{m_1})$ and lone variables $\hat{\mathbf{r}} = (\hat{r}_1, ..., \hat{r}_{m_2})$. Specifically, the polynomial k_i is expressed as

$$k_i = \sum_{j\in[n_\alpha]} \delta_{1,i,j}\alpha_j \;+\; \sum_{j\in[n_\beta]} \delta_{2,i,j}\beta_j \;+\; \sum_{j\in[m_2]} \hat{\delta}_{i,j}\hat{r}_j \;+\; \sum_{\substack{j\in[m_1]\\ k\in[n_b]}} \delta_{3,i,j,k} r_j b_k,$$

  for all $i \in [m_3]$, where $\delta_{1,i,j}, \delta_{2,i,j}, \hat{\delta}_{i,j}, \delta_{3,i,j,k} \in \mathbb{Z}_p$.
- $\text{EncCt}(x) \to (w_1, w_2, w_2', c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \mathbf{c}''(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta}))$: On input $x \in \mathcal{X}$, this algorithm outputs a blinding variable c_M and three vectors of polynomials $\mathbf{c} = (c_1, ..., c_{w_3})$, $\mathbf{c}' = (c_1', ..., c_{w_4}')$ and $\mathbf{c}'' = (c_1'', ..., c_{w_5}'')$ defined over non-lone variables $\mathbf{s} = (s = s_0, s_1, s_2, ..., s_{w_1})$, lone variables $\hat{\mathbf{s}} = (\hat{s}_1, ..., \hat{s}_{w_2})$ and special lone variables $\tilde{\mathbf{s}} = (\tilde{s}_1, ..., \tilde{s}_{w_2'})$. Specifically, the polynomial c_i is expressed as

$$c_i = \sum_{\substack{j\in[w_1]\\ k\in[n_b]}} \eta_{1,i,j,k} s_j b_k \;+\; \sum_{j\in[w_2]} \hat{\eta}_{1,i,j}\hat{s}_j$$

for all $i \in [w_3]$, where $\eta_{1,i,j,k}, \hat{\eta}_{1,i,j} \in \mathbb{Z}_p$, the polynomial c'_i is expressed as

$$c'_i = \sum_{\substack{j \in [n_\alpha] \\ j' \in \overline{[w_1]}}} \eta'_{1,i,j,j'} \alpha_j s_{j'} \; + \sum_{\substack{j \in [n_\beta] \\ j' \in \overline{[w_1]}}} \eta'_{2,i,j,j'} \beta_j s_{j'} \; + \sum_{j \in [w'_2]} \hat{\eta}'_{i,j} \tilde{s}_j \; ,$$

for all $i \in [w_4]$, where $\eta'_{1,i,j,j'}, \eta'_{2,i,j,j'}, \hat{\eta}'_{i,j} \in \mathbb{Z}_p$, the polynomial c''_i is expressed as

$$c''_i = \sum_{\substack{j \in \overline{[w_1]} \\ k \in [n_\beta]}} \eta_{2,i,j,k} s_j \beta_k \; + \sum_{j \in [w'_2]} \hat{\eta}_{2,i,j} \tilde{s}_j,$$

for all $i \in [w_5]$, where $\eta_{2,i,j,k}, \hat{\eta}_{2,i,j} \in \mathbb{Z}_p$, and the polynomial c_M is expressed as

$$c_M = \sum_{j \in [w'_2]} \zeta_j \tilde{s}_j \; + \sum_{\substack{j \in [n_\alpha] \\ j' \in \overline{[w_1]}}} \zeta_{1,j,j'} \alpha_j s_{j'} \; + \sum_{\substack{j \in [n_\beta] \\ j' \in \overline{[w_1]}}} \zeta_{2,j,j'} \beta_j s_{j'} \; ,$$

where $\zeta_j, \zeta_{1,j,j'}, \zeta_{2,j,j'} \in \mathbb{Z}_p$.

- $\mathrm{Pair}(x, y) \to (\mathbf{e}', \mathbf{e}'', \mathbf{E}, \overline{\mathbf{E}})$: On input x, and y, this algorithm outputs two vectors $\mathbf{e}' \in \mathbb{Z}_p^{w_4}, \mathbf{e}'' \in \mathbb{Z}_p^{w_5}$ and two matrices $\mathbf{E}$ and $\overline{\mathbf{E}}$ of sizes $(w_1 + 1) \times m_3$ and $w_3 \times m_1$, respectively.

A PES-ISA is correct if, for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ such that $P(x, y) = 1$, it holds that $\mathbf{e}'\mathbf{c}'^{\intercal} + \mathbf{e}''\mathbf{c}''^{\intercal} + \mathbf{s}\mathbf{E}\mathbf{k}^{\intercal} + \mathbf{c}\overline{\mathbf{E}}\mathbf{r}^{\intercal} = c_M$.

Remark 1. PES-AC17 and PES-Ven23 are subclasses of our class of PES. In particular, schemes with $\boldsymbol{\alpha} = \alpha$, $\boldsymbol{\beta} = \emptyset$, $c_M = \alpha s$ and $\mathbf{c}' = \mathbf{c}'' = \emptyset$ are also PES-AC17 and schemes with $\boldsymbol{\beta} = \mathbf{c}'' = \emptyset$ are PES-Ven23.

3.1 The special symbolic property

We introduce a new definition for the special selective symbolic property. Concretely, we formalize the behavior of the predicate after the corruption of variables.

Corruptable variables and the predicate. Our definition of the symbolic property supports the corruption of variables by listing a set of corruptable variables. Roughly, the special symbolic property can be invoked in the security proof, as long as it has been proven for a set of corruptable variables that contains the subset of corrupted variables in the security game. Because corruption of the variables often impacts the satisfiability of the predicate P, we also include a predicate P_{cor} that is related to the original predicate P and the corruptable variables. In particular, P_{cor}, with $\mathsf{cor} = (\Gamma_{\text{PES-ISA}}, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b})$, expands the predicate P for scheme $\Gamma_{\text{PES-ISA}}$ with respect to the corruptable variables. For example, suppose that the keys are generated by two authorities, and the first authority manages attribute att_1 and the second one manages att_2. If none of the authorities' keys are corrupted, then the predicate P evaluates true on the policy $\mathsf{att}_1 \wedge \mathsf{att}_2$ on any set that contains both att_1 and att_2. However, if the first authority's keys are corrupted, then any set containing att_2 evaluates to true, because the first authority can always generate a key for att_1 if needed. The predicate P_{cor} expresses this expansion of the predicate P.

Definition 4 (Special selective symbolic property (SSSP) for PES-ISA)**.** A scheme $\Gamma_{\text{PES-ISA}} = (\mathrm{Param}, \mathrm{EncKey}, \mathrm{EncCt}, \mathrm{Pair})$ for a predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ satisfies the (d_1, d_2)-selective symbolic property for positive integers d_1 and d_2 and indices of corruptable variables $\mathfrak{a}_1 \subsetneq [n_\alpha]$, $\mathfrak{a}_2 \subsetneq [n_\beta]$, $\mathfrak{b} \subsetneq [n_b]$ and the predicate P_{cor} that expands the predicate P with respect to the corruptable variables, if there exist deterministic polynomial-time algorithms EncB, EncS, and EncR such that for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ with $P_{\mathsf{cor}}(x, y) = 0$, such that we have that

- $\mathrm{EncB}(x, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}) \to \mathbf{a}_1, ..., \mathbf{a}_{n_\alpha}, \mathbf{b}_1, ..., \mathbf{b}_{n_\beta} \in \mathbb{Z}_p^{d_1}, \mathbf{B}_1, ..., \mathbf{B}_{n_b} \in \mathbb{Z}_p^{d_1 \times d_2}$;
- $\mathrm{EncR}(x, y) \to \mathbf{r}_1, ..., \mathbf{r}_{m_1} \in \mathbb{Z}_p^{d_2}, \hat{\mathbf{r}}_1, ..., \hat{\mathbf{r}}_{m_2} \in \mathbb{Z}_p^{d_1}$;
- $\mathrm{EncS}(x) \to \mathbf{s}_0, ..., \mathbf{s}_{w_1} \in \mathbb{Z}_p^{d_1}, \hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_{w_2} \in \mathbb{Z}_p^{d_2}, \tilde{\mathbf{s}}_1, ..., \tilde{\mathbf{s}}_{w_2'} \in \mathbb{Z}_p$;

such that, if we substitute

$$\hat{s}_{i'} : \hat{\mathbf{s}}_{i'} \quad \tilde{s}_{i''} : \tilde{\mathbf{s}}_{i''} \quad s_i b_j : \mathbf{s}_i \mathbf{B}_j \quad \alpha_l : \mathbf{a}_l^\intercal \quad \beta_{l'} : \mathbf{b}_{l'}^\intercal$$
$$\alpha_l s_i : \mathbf{s}_i \mathbf{a}_l^\intercal \quad \beta_{l'} s_i : \mathbf{s}_i \mathbf{b}_{l'}^\intercal \quad \hat{r}_{k'} : \hat{\mathbf{r}}_{k'}^\intercal \quad r_k b_j : \mathbf{B}_j \mathbf{r}_k^\intercal,$$

for $i \in [w_1]$, $i' \in [w_2]$, $i'' \in [w_2']$, $j \in [n_b]$, $k \in [m_1]$, $k' \in [m_2]$, $l \in [n_\alpha]$, $l' \in [n_\beta]$ in all the polynomials of $\mathbf{k}$ and $\mathbf{c}$, $\mathbf{c}'$ and $\mathbf{c}''$ (output by EncKey and EncCt, respectively), they evaluate to $\mathbf{0}^{d_1}, \mathbf{0}^{d_2}, 0, 0$ and 0, respectively, and the polynomial c_M to nonzero. Furthermore,

- for $j \in \mathfrak{a}_1, j' \in \mathfrak{a}_2$, we have $\mathbf{a}_j = \mathbf{b}_{j'} = \mathbf{0}^{d_1}$;
- and for $j \in \mathfrak{b}$, we have that $\mathbf{B}_j = \mathbf{0}^{d_1 \times d_2}$.

Remark 2. Because the substitution vectors in the SSSP proofs are the same for the $\mathbf{c}'$ and $\mathbf{c}''$ parts (as well as the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ parts), we can simply move all polynomials in $\mathbf{c}''$ to $\mathbf{c}'$ for the security proofs, and split the two before instantiation with our compiler.

3.2 Multi-key and multi-ciphertext security

We also give more general security definitions for PES-ISA that we use as a foundation for the security proofs in the generic group model. Our multi-key and multi-ciphertext definitions are derived from the notion of trivially broken by Agrawal and Chase [AC17b] and the notions of trivial and collusion security in [dlPVA23]. We start by recalling trivial security which we then strengthen to multi-key and multi-ciphertext security.

Definition 5 (Trivial security for PES-ISA)**.** We call a PES-ISA for predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ trivially secure if, for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ with $P(x,y) = 0$, $(w_1, w_2, w_2', c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \boldsymbol{\beta}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}, \boldsymbol{\beta})) \leftarrow \mathrm{EncCt}(x)$, and $(m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b})) \leftarrow \mathrm{EncKey}(y)$, it holds that

$$\mathbf{e}'\mathbf{c}'^\intercal + \mathbf{e}''\mathbf{c}''^\intercal + \mathbf{s}\mathbf{E}\mathbf{k}^\intercal + \mathbf{c}\overline{\mathbf{E}}\mathbf{r}^\intercal \neq c_M,$$

for all $(\mathbf{e}', \mathbf{e}'', \mathbf{E}, \overline{\mathbf{E}}) \in (\mathbb{Z}_p^{w_4}, \mathbb{Z}_p^{w_5}, \mathbb{Z}_p^{(w_1+1)\times m_3}, \mathbb{Z}_p^{w_3 \times m_1})$. This is equivalent to stating that the span consisting of $\mathbf{c}'$, $\mathbf{c}''$, all combinations of $\mathbf{s}$ and $\mathbf{k}$ and all combinations of $\mathbf{c}$ and $\mathbf{r}$ does not contain c_M.

Definition 6 (Multi-key and multi-ciphertext (MK-MC) security)**.** We call a PES-ISA for predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ multi-key and multi-ciphertext secure if, for all $n_k, n_c = \mathsf{poly}(\lambda)$, $x_1, ..., x_{n_c} \in \mathcal{X}$ and $y_1, ..., y_{n_k} \in \mathcal{Y}$ with $P(x_i, y_j) = 0$ and

$$\mathbf{c}_{x_{1,\ldots,n_c}},\ \mathbf{c}'_{x_{1,\ldots,n_c}},\ \mathbf{c}''_{x_{1,\ldots,n_c}},\ \mathbf{s}_{x_{1,\ldots,n_c}},$$
$$\mathbf{c_M} = (c_{M,x_1}, ..., c_{M,x_{n_c}}),\ \mathbf{k}_{y_{1,\ldots,n_k}},\ \mathbf{r}_{y_{1,\ldots,n_k}},$$

obtained by running $\mathrm{EncCt}(x_i)$, and $\mathrm{EncKey}(y_j)$ for all i and j and concatenating the encodings, we have

$$\mathbf{e}'(\mathbf{c}'_{x_{1,\ldots,n_c}})^\intercal + \mathbf{e}''(\mathbf{c}''_{x_{1,\ldots,n_c}})^\intercal$$
$$+\mathbf{s}_{x_{1,\ldots,n_c}}\mathbf{E}\mathbf{k}^\intercal_{y_{1,\ldots,n_k}} + \mathbf{c}_{x_{1,\ldots,n_c}}\overline{\mathbf{E}}\mathbf{r}^\intercal_{y_{1,\ldots,n_k}} \neq \mathbf{e_M}\mathbf{c_M}^\intercal,$$

for all $\mathbf{e}'$, $\mathbf{e}''$, $\mathbf{E}$, $\overline{\mathbf{E}}$ and $\mathbf{e_M}$ (each with entries in $\mathbb{Z}_p$).

We show that considering the listed combinations of key and ciphertext encodings is sufficient to argue security of PES-ISA. The reason is that products of other combinations, e.g., common variables and key encodings, are not helping an attacker break the scheme. We prove this in Section 3.3.

Definition 7 (Strong MK-MC security)**.** We call a PES-ISA for predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ *strongly* multi-key and multi-ciphertext secure if, for all $n_k, n_c = \mathsf{poly}(\lambda)$, $x_1, ..., x_{n_c} \in \mathcal{X}$ and $y_1, ..., y_{n_k} \in \mathcal{Y}$ with $P(x_i, y_j) = 0$ and

$$\mathbf{c}_{x_{1,\dots,n_c}},\ \mathbf{c}'_{x_{1,\dots,n_c}},\ \mathbf{c}''_{x_{1,\dots,n_c}},\ \mathbf{s}_{x_{1,\dots,n_c}},$$
$$\mathbf{c_M} = (c_{M,x_1}, ..., c_{M,x_{n_c}}),\ \mathbf{k}_{y_{1,\dots,n_k}},\ \mathbf{r}_{y_{1,\dots,n_k}},$$

obtained by running $\mathrm{EncCt}(x_i)$, and $\mathrm{EncKey}(y_j)$ for all i and j and concatenating the encodings, we have

$$\mathbf{e} \cdot \mathbf{p_{all}}^\intercal \neq \mathbf{e_M} \mathbf{c_M}^\intercal,$$

for all $\mathbf{e} \in \mathbb{Z}_p^{|\mathbf{p_{all}}|}$, where

$$\mathbf{p_{all}} = \boldsymbol{\alpha} \| \mathbf{c}'_{x_{1,\dots,n_c}} \| (\mathbf{p}_{\mathsf{combis}} \setminus \mathbf{p}_{\mathsf{special}}),$$

where

$$\mathbf{p}_{\mathsf{combis}} = \{p_i p_j \mid p_i, p_j \in (1, \boldsymbol{\beta}, \mathbf{b}, \mathbf{s}_{x_{1,\dots,n_c}}, \mathbf{k}_{y_{1,\dots,n_k}}, \mathbf{c}_{x_{1,\dots,n_c}}, \mathbf{c}''_{x_{1,\dots,n_c}}, \mathbf{r}_{y_{1,\dots,n_k}})\}$$

consists of all combinations that can be made and

$$\mathbf{p}_{\mathsf{special}} = \{s_j \beta_k \mid \forall i \in [n_c], s_j \in \mathbf{s}_{x_i}, \beta_k \in \boldsymbol{\beta}\ [\ \exists i' \in [m_3]\ [\ \eta_{2,i',j,k} \neq 0 \vee \eta'_{2,i',j,k} \neq 0\ ]\ ]\ \}$$

consists of all combinations $s_j \beta_k$ that do not occur in the ciphertext polynomials.

Note that the $\mathbf{p}_{\mathsf{special}}$ combinations are excluded from the combinations because they cannot be created once instantiated in the pairing groups with the restrictions we will pose on the distributions (cf. Section 4.1).

3.3 Security notions in matrix notation

At the core of our security results, we analyze, formalize and systematize the relationship between the special selective symbolic property and the strong MK-MC security for our class of schemes. In this way, we can use SSSP to derive security results in multiple security models. We achieve this by showing that SSSP implies the strong MK-MC security notion. To prove this, we use the matrix notation proposed in ACABELLA [dlPVA23], which is used to show that their variant of SSSP implies multi-key security (for one ciphertext) for the AC17 class of PES. Roughly, the matrix notation allows us to use the strong relationship between the (column) kernel and the row span of the matrix to argue about the relationship between SSSP (which is strongly related to the kernel) and strong MK-MC security (which is strongly related to the row span).

Definition 8 (Matrix notations for all key-ciphertext combinations)**.** For any PES-ISA for predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, we define the vector

$$\mathbf{p_{enc}}_{x,y} \coloneqq \left(\{(\mathbf{c}')_i\}_{i \in [w_4]} \cup \{(\mathbf{c}'')_i\}_{i \in [w_5]} \cup \{s_j k_i\}_{\substack{j \in [w_1] \\ i \in [m_3]}} \cup \{r_j c_i\}_{\substack{j \in [m_1] \\ i \in [w_3]}} \right),$$

for all $x \in \mathcal{X}, y \in \mathcal{Y}$, as well as its matrix decomposition

$$\mathbf{p_{enc}}_{x,y} = \mathbf{M}_{x,y} \cdot \mathbf{v}^\intercal_{x,y},$$

so that the entries of $\mathbf{M}_{x,y}$ consist of the coefficients

$$\delta_{1,i,j},\ \delta_{2,i,j},\ \hat{\delta}_{i,j},\ \delta_{3,i,j,k},\ \zeta_j,\ \zeta_{1,j,j'},\ \zeta_{2,j,j'}\ ,$$
$$\eta_{1,i,j,k},\ \hat{\eta}_{1,i,j},\ \eta_{2,i,j,k},\ \hat{\eta}_{2,i,j},\ \eta'_{1,i,j,j'},\ \eta'_{2,i,j,j'},\ \hat{\eta}'_{i,j}\ ,$$

and the entries of $\mathbf{v}_{x,y}$ consist of monomials

$$\alpha_j s_{j'},\ \beta_j s_{j'},\ \tilde{s}_{j'},\ \hat{r}_j s_{j'},\ r_j s_{j'} b_k,\ r_j \hat{s}_{j'}\ .$$

Both $\mathbf{p_{enc}}_{x,y}$ and $(\mathbf{M}_{x,y}, \mathbf{v}_{x,y})$ represent all key-ciphertext combinations considered in the trivial security security definitions.

Lemma 1. *Let $\mathbf{p_{enc}}_{x,y}$ be as in Definition 8. Then, $\mathbf{p_{enc}}_{x,y}$ represents all key-ciphertext combinations considered in the trivial and multi-key and multi-ciphertext security definitions, i.e.,*

$$\{\mathbf{e}'\mathbf{c}'^{\intercal} + \mathbf{e}''\mathbf{c}''^{\intercal} + \mathbf{s}\mathbf{E}\mathbf{k}^{\intercal} + \mathbf{c}\overline{\mathbf{E}}\mathbf{r}^{\intercal} \mid (\mathbf{e}', \mathbf{e}'', \mathbf{E}, \overline{\mathbf{E}}) \in (\mathbb{Z}_p^{w_4}, \mathbb{Z}_p^{w_5}, \mathbb{Z}_p^{(w_1+1)\times m_3}, \mathbb{Z}_p^{w_3\times m_1})\}$$
$$= \{\mathbf{e}\mathbf{p_{enc}}_{x,y}^{\intercal} \mid \mathbf{e} \in \mathbb{Z}_p^{w_4+w_5+(w_1+1)m_3+w_3m_1}\}$$

Proof. The proof in Section 3.2 of [dlPVA23] generalizes naturally to our class of schemes. □

In ACABELLA [dlPVA23, §3.2], it is proven that finding a linear combination for $\mathbf{p_{enc}}_{x,y}$ that yields some polynomial tv is equivalent to finding a linear combination of the rows of $\mathbf{M}_{x,y}$ that yields the vector $\mathbf{tv}$ such that $\mathsf{tv} = \mathbf{tv} \cdot \mathbf{v}_{x,y}^{\intercal}$.

Lemma 2. *Let $\mathbf{p_{enc}}_{x,y}$ and $(\mathbf{M}_{x,y}, \mathbf{v}_{x,y})$ be as in Definition 8, let tv be some polynomial and $\mathsf{tv} = \mathbf{tv} \cdot \mathbf{v}_{x,y}^{\intercal}$ be its decomposition with respect to the vector $\mathbf{v}_{x,y}$. Then the following two statements are equivalent:*

(i) There exists some $\mathbf{e} \in \mathbb{Z}_p^{w_4+w_5+(w_1+1)m_3+w_3m_1}$ such that $\mathbf{e} \cdot \mathbf{p_{enc}}_{x,y}^{\intercal} = \mathsf{tv}$;

(ii) There exists some $\mathbf{e} \in \mathbb{Z}_p^{w_4+w_5+(w_1+1)m_3+w_3m_1}$ such that $\mathbf{e} \cdot \mathbf{M}_{x,y}^{\intercal} = \mathbf{tv}$.

To show that a vector is *not* in the span of a matrix, we use an important result from linear algebra, which has been frequently used in the field of ABE [GPSW06b, §A] (and has been proven in many works before it):

Proposition 1. *A vector $\mathbf{tv}$ is not in the span of the rows of $\mathbf{M}$ if and only if there exists a vector $\mathbf{w}$ such that $\mathbf{M} \cdot \mathbf{w}^{\intercal} = \mathbf{0}^{\intercal}$ and $\mathsf{tv} \cdot \mathbf{w}^{\intercal} \neq 0$.*

Although we can use this proposition to prove trivial security (i.e., by considering the key-ciphertext combinations in matrix notation and subsequently showing that some vector $\mathbf{w}$ exists), we cannot use it to prove strong trivial security, which requires that a span of vectors cannot overlap the span of the rows of $\mathbf{M}$. Following a similar approach as ACABELLA, we strengthen the above definition to consider vectors $\mathbf{w}$ of a certain structure. This structure is strongly related to the properties we require from the vectors and matrices in the special selective symbolic property (Definition 4). In fact, we show later that the vector $\mathbf{w}$ can be directly used to construct a special selective symbolic property proof.

Definition 9 (Kernel property for MK-MC security)**.** Let $x \in \mathcal{X}$ and consider for all $y \in \mathcal{Y}$ with $P(x, y) = 0$ the matrix decomposition $\mathbf{p_{enc}}_{x,y} = \mathbf{M}_{x,y} \cdot \mathbf{v}_{x,y}^{\intercal}$ for some PES-ISA, where $\mathbf{v}_{x,y} = \mathbf{v}_{x,y,1} \| \mathbf{v}_{x,y,2}$ is such that the entries of $\mathbf{v}_{x,y,1}$ are of the form $\alpha_j s_{j'}, \beta_j s_{j'}$ and $\tilde{s}_{j'}$ and the entries of $\mathbf{v}_{x,y,2}$ are of the form $\hat{r}_j s_{j'}, r_j s_{j'} b_k, r_j \hat{s}_{j'}$. Let $\mathsf{tv} = \mathbf{tv}_{x,y} \cdot \mathbf{v}_{x,y}^{\intercal}$ be the target vector with respect to $\mathbf{v}_{x,y}$. If, for all $y \in \mathcal{Y}$, there exist $\mathbf{w}_{x,y} = \mathbf{w}_{x,y,1} \| \mathbf{w}_{x,y,2}$ (with $|\mathbf{w}_{x,y,1}| = |\mathbf{v}_{x,y,2}|$) such that

- $\mathbf{M}_{x,y} \cdot \mathbf{w}_{x,y}^\intercal = \mathbf{0}$ for all $y \in \mathcal{Y}$;
- $\mathbf{tv}_{x,y} \cdot \mathbf{w}_{x,y}^\intercal \neq 0$ for all $y \in \mathcal{Y}$;
- $\mathbf{w}_{x,y,1} = \mathbf{w}_{x,y',1}$ for all $y, y' \in \mathcal{Y}$,

then we say that the PES-ISA satisfies the kernel property for multi-key and multi-ciphertext security.

Proposition 2. *Consider a PES-ISA with the kernel property for multi-key and multi-ciphertext security (Definition 9). Then the PES-ISA is multi-key and multi-ciphertext secure (Definition 6).*

Proof. The proof is almost identical to the proof of Theorem 2 in ACABELLA [dlPVA23], except that we also need to prove multi-ciphertext security, and we only require that the "$\mathbf{v}_{x,y,1}$-part" of $\mathbf{w}_{x,y}$, i.e., $\mathbf{w}_{x,y,1}$, is the same for each x, y. First, we show that we can construct a matrix decomposition $\mathbf{M} \cdot \mathbf{v}^\intercal$ from the matrix decompositions $\mathbf{M}_{x,y} \cdot \mathbf{v}_{x,y}^\intercal$, as well as the associated target vector $\mathbf{tv}$ from $\mathbf{tv}_{x,y}$. (Note that $\mathbf{v}_{x,y}$ and $\mathbf{v}_{x',y}$ are completely disjoint.) Analogously, we can construct a vector $\mathbf{w}$ in the kernel of $\mathbf{M}$, from the vectors $\mathbf{w}_{x,y}$ for which it holds that $\mathbf{tv} \cdot \mathbf{w}^\intercal \neq 0$. This can be done in the same way as in ACABELLA. The only distinction is that we have to prove that we cannot construct any linear combination of c_{M,x_i}. To do this, we argue that the vector $\mathbf{w}$ that we construct from all $\mathbf{w}_{x,y}$ can be used to construct a vector $\mathbf{w}_{x_i}$ that proves that there exists no linear combination of $c_{M,x_1,...,x_{n_c}}$ in which c_{M,x_i} is used, i.e., has a nonzero coefficient. We construct $\mathbf{w}_{x_i}$ from $\mathbf{w}$ by setting the entries associated with $\mathbf{v}_{x_{i'},y_j}$ for all $i' \neq i$ and j to zero. We then have $\mathbf{tv} \cdot \mathbf{w}_{x_i}^\intercal = \mathbf{tv}_{x_i,y} \cdot \mathbf{w}_{x_i,y}^\intercal \neq 0$. In particular, if we decompose $c_{M,x_{i'}}$ with respect to $\mathbf{v}$ for all $i' \neq i$, i.e., $\mathbf{tv}_{x_i'} \cdot \mathbf{v}^\intercal = c_{M,x_{i'}}$, and compute any linear combination of $\mathbf{tv}_{x_i'}$ and add it to $\mathbf{tv}$, obtaining $\mathbf{tv}'$, then we have $\mathbf{tv}' \cdot \mathbf{w}_{x_i}^\intercal = \mathbf{tv} \cdot \mathbf{w}_{x_i}^\intercal \neq 0$. Because $\mathbf{tv} \cdot \mathbf{v}^\intercal = \sum_{i \in [n_c]} c_{M,x_i}$, it therefore holds that $\mathbf{tv}'$ is the decomposition of any linear combination of $c_{M,x_1}, ..., c_{M,x_{n_c}}$ for which the coefficient for c_{M,x_i} is 1. By Proposition 1, it then follows that $\mathbf{tv}'$ is not in the span of $\mathbf{M}$ (for any linear combination, for any x_i). Hence, there is no linear combination of $c_{M,x_1}, ..., c_{M,x_{n_c}}$ in the span, and therefore, it follows with Lemma 2 that the scheme is MK-MC secure. □

We also show that other combinations, e.g., key polynomials with key polynomials, do not affect the security of the PES.

Proposition 3 (Completeness of the kernel property for MK-MC)**.** *Consider a PES-ISA that satisfies the kernel property for multi-key and multi-ciphertext security, and let* $\{p_1, ..., p_n\}$ *be any set of polynomials over the variables* $\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b}, \mathbf{r}, \hat{\mathbf{r}}, \mathbf{s}, \hat{\mathbf{s}}, \tilde{\mathbf{s}}$ *and coefficients in* $\mathbb{Z}_p$*. If none of the monomials of all* p_i *occurs in any of the polynomials in* $\mathbf{penc}_{x,y}$ *for all* $x \in \mathcal{X}, y \in \mathcal{Y}$*, then* $\{p_i\}_i$ *does not affect the security of the PES-ISA. More formally, we say that polynomial* $\{p_i\}_i$ *do not affect the MK-MC security of the PES-ISA if*

$$\mathbf{e} \cdot \left(\mathbf{penc}_{x,y} \cup \{p_i\}_{i \in [n]}\right) \neq \mathbf{e_M} \mathbf{c_M}^\intercal$$

for all $\mathbf{e} \in \mathbb{Z}_p^{|\mathbf{penc}_{x,y}|+n}$ *and* $\mathbf{e_M} \in \mathbb{Z}_p^{|\mathbf{c_M}|}$.

Proof. Let $\mathbf{M}_{x,y} \cdot \mathbf{v}_{x,y}^\intercal$ denote the matrix decomposition of $\mathbf{penc}_{x,y}$ and let $\mathbf{M}_{p_1,...,p_n} \cdot \mathbf{v}_{p_1,...,p_n}^\intercal$ denote the matrix decomposition of $(\{p_i\}_{i \in [n]})$. Then, because none of the monomials in $p_1, ..., p_n$ occur in $\mathbf{penc}_{x,y}$, we know that $\mathbf{v}_{x,y}$ and $\mathbf{v}_{p_1,...,p_n}$ are disjoint. We can then create a matrix decomposition for $(\mathbf{penc}_{x,y} \cup \{p_i\}_{i \in [n]})$ as

$$\begin{pmatrix} \mathbf{M}_{x,y} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{p_1,...,p_n} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_{x,y}^\intercal \\ \mathbf{v}_{p_1,...,p_n}^\intercal \end{pmatrix},$$

and similarly, we can create some vector $\mathbf{w}'$ by taking $\mathbf{w}_{x,y}$ and appending $|\mathbf{v}_{p_1,\ldots,p_n}|$ zeros. This is then a proof for the kernel property for MK-MC security as in Definition 9, and thus, the scheme is MK-MC secure. □

Via this proposition, we can prove that none of the combinations in $\mathbf{p}_{\mathbf{all}}$ (that are not the combinations of the form $\mathbf{p}_{\mathbf{enc}_{x,y}}$) affect the MK-MC security of the scheme, and therefore, that a PES-ISA that satisfies the kernel property fro MK-MC security is strongly MK-MC secure.

Lemma 3. *Consider a PES-ISA that satisfies the kernel property for MK-MC security, and let the polynomial p_i be any of the entries in $\mathbf{p}_{\mathbf{all}}$ (Definition 7) that does not occur in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$ (for all $i \in [n_c], j \in [n_k]$). Then p_i does not affect the security of the PES-ISA.*

Proof. We can apply Proposition 3 to prove this. To invoke that proposition, we have to show that none of the polynomials in $\mathbf{p}_{\mathbf{all}}$ that do not occur in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$ (for all $i \in [n_c], j \in [n_k]$)) have any monomials that also occur in the polynomials in $\mathbf{p}_{\mathbf{all}}$. To show this, we first observe that the monomials in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$ are of the form:

$$\alpha_j s_{j'}, \quad \beta_j s_{j'}, \quad \tilde{s}_{j'}, \quad \hat{r}_j s_{j'}, \quad r_j s_{j'} b_k, \quad r_j \hat{s}_{j'} \ .$$

For the other polynomials in $\mathbf{p}_{\mathbf{all}}$, we can see that the monomials that occur in $\boldsymbol{\alpha}$ do not occur in the list above. For the other polynomial combinations, we argue with what monomials they need to be combined in order to obtain a monomial that occurs in the list above:

- β_j would need to be paired with $s_{j'}$, and these combinations do exist in $\mathbf{p}_{\mathbf{all}}$, but those combinations of β_j and $s_{j'}$ that do occur in the above polynomials are filtered out with $\mathbf{p}_{\mathsf{special}}$;
- b_k would need to be paired with $r_j s_{j'}$, which is not a combination that occurs;
- $s_{j'}$ would need to be paired with α_j, β_j, $\hat{r}_j$ or $r_j b_k$, but these only occur in the key polynomials, and these pairs are in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$;
- k_i would need to be paired with $s_{j'}$, but these pairs are in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$;
- c_i would need to be paired with r_j, but these pairs are in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$;
- c_i'' does not need to be pair, but is part of $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$;
- r_j would need to be paired with $s_{j'} b_k$ or $\tilde{s}_{j'}$, but these occur only in the ciphertext polynomials, and these pairs are in $\mathbf{p}_{\mathbf{enc}_{x_i,y_j}}$.

Thus, these polynomials do not affect the security. □

Corollary 1. *A PES-ISA that satisfies the kernel property for MK-MC security is strongly MK-MC secure.*

Discussion on the MK-MC security property in FABEO. We prove strong MK-MC security, which is a property for multiple keys and multiple ciphertexts, from the kernel property for MK-MC security, which is a property for one key and one ciphertext. This approach is similar to that in FABEO [RW22], in that we also prove security for multiple keys and multiple ciphertexts from a security notion for one key and one ciphertext. However, the crucial difference is in that our single-key and single-ciphertext property considers the *kernel* of the matrix $\mathbf{M}_{x,y}$, whereas FABEO considers its *span*. Additionally, we consider the *existence* of some vector in the kernel, whereas FABEO considers the

absence of any linear combination of the vectors corresponding to αs_j (i.e., the products that do not depend on the randomness $\mathbf{r}, \hat{\mathbf{r}}$ introduced in the key).

Although ACABELLA gives a kernel property that implies the span property required in FABEO, it requires essentially an SSSP proof for each product αs_j (including αs itself). This seems to suggest that the span property in FABEO is significantly stronger than the kernel property. However, note that ACABELLA does not prove this. In fact, all the schemes analyzed with the ACABELLA tool satisfy both the weaker kernel property and the (seemingly stronger) span property, which might mean that, e.g., an SSSP proof for αs implies an SSSP proof for any product αs_j. (Note, however, that this is also not proven in ACABELLA.) It is thus interesting to briefly consider whether a similar span property could be leveraged as a starting point to prove strong multi-key and multi-ciphertext security for PES-ISA.

We argue that a similar span property would be strictly stronger for PES-ISA. A natural generalization of the span property for PES-AC17 to the class of PES-ISA would disallow linear combinations of the vectors associated with $\alpha_i s_j$, $\beta_i s_j$ and $\tilde{s}_j$. We can show that this property is too strong for most (if not all) multi-authority ABE schemes. In particular, the ciphertext polynomials $\mathbf{c}'$ and $\mathbf{c}''$ are linear combinations of these monomials, and thus, any scheme with $\mathbf{c}', \mathbf{c}'' \neq \emptyset$ (like most existing multi-authority schemes) would not satisfy the property. Possibly, there exist less straightforward generalizations, e.g., by considering linear combinations of $\alpha_i s_j$, $\beta_i s_j$ and $\tilde{s}_j$ that contains at least one monomial that occurs in c_M. However, this specific relaxation does not hold for the PESs implied by most existing multi-authority schemes either, because $c_M = \tilde{s}$ and there exists at least one polynomial in $\mathbf{c}'$ and $\mathbf{c}''$ in which $\tilde{s}$ occurs with a nonzero coefficient.

3.4 Handling corruption

To prove security against corruptions, we extend the results for the case without corruptions. To this end, we introduce the notion of corruption-induced key and ciphertext encodings. These replace the master-key, semi-common and common variables that are corrupted for integers. For example, if one of the key encoding polynomials is $\alpha + rb$, and b is corrupted, the corruption-induced key encoding polynomial would sample an integer for b and move it to the coefficient space. Note that this also changes the structure of the polynomials, and therefore expands the potential combinations that can be made in an attack.

Definition 10 (Corruption-induced key and ciphertext encodings)**.** Consider a PES-ISA as defined in Definition 3 and corruptable variables $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$. Let $\mathsf{Ev}_\alpha \colon \mathfrak{a}_1 \to \mathbb{Z}_p$, $\mathsf{Ev}_\beta \colon \mathfrak{a}_2 \to \mathbb{Z}_p$ and $\mathsf{Ev}_b \colon \mathfrak{b} \to \mathbb{Z}_p$ be evaluation functions that map all corruptable variables to integers. Let $\boldsymbol{\alpha}_{\mathsf{hon}}, \boldsymbol{\beta}_{\mathsf{hon}}, \mathbf{b}_{\mathsf{hon}}$ denote the subsets of master-key, semi-common and common variables $\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b}$ that are not corrupted. We define the corruption-induced key and ciphertext encodings as follows.

- $\mathrm{CorruptKey}(y, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}, \mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b) \to \mathbf{k}_{\mathsf{cor}}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}_{\mathsf{hon}}, \boldsymbol{\beta}_{\mathsf{hon}}, \mathbf{b}_{\mathsf{hon}})$: On input $y \in \mathcal{Y}$, corruptable variables $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$ and its evaluation functions $\mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b$, this algorithm first runs $(m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b})) \leftarrow \mathrm{EncKey}(y)$ and then outputs a vector of corruption-induced key polynomials $\mathbf{k}_{\mathsf{cor}} = (\check{k}_{\mathsf{cor},1}, ..., \check{k}_{\mathsf{cor},m_3})$, where

$$\check{k}_{\mathsf{cor},i} = \delta_{0,i} + \sum_{j \in [n_\alpha] \setminus \mathfrak{a}_1} \delta_{1,i,j} \alpha_j + \sum_{j \in [n_\beta] \setminus \mathfrak{a}_2} \delta_{2,i,j} \beta_j + \sum_{j \in [m_2]} \hat{\delta}_{i,j} \hat{r}_j + \sum_{\substack{j \in [m_1] \\ k \in [n_b] \setminus \mathfrak{b}}} \delta_{3,i,j,k} r_j b_k + \sum_{\substack{j \in [m_1] \\ k \in \mathfrak{b}}} \check{\delta}_{3,i,j,k} r_j \ ,$$

where $\delta_{0,i} = \sum_{j \in \mathfrak{a}_1} \delta_{1,i,j} \mathsf{Ev}_\alpha(j) + \sum_{j \in \mathfrak{a}_2} \delta_{2,i,j} \mathsf{Ev}_\beta(j)$ and $\check{\delta}_{3,i,j,k} = \delta_{3,i,j,k} \mathsf{Ev}_b(k)$.

- $\mathsf{CorruptCt}(x, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}, \mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b) \to (c_M, \mathbf{c}_{\mathsf{cor}}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}_{\mathsf{hon}}), \mathbf{c}'_{\mathsf{cor}}(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}_{\mathsf{hon}}, \boldsymbol{\beta}_{\mathsf{hon}}), \mathbf{c}''_{\mathsf{cor}}(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta}_{\mathsf{hon}}))$: On input $x \in \mathcal{X}$, corruptable variables $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$ and its corresponding evaluation functions $\mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b$, this algorithm first runs $(w_1, w_2, w'_2, c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \mathbf{c}''(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta})) \leftarrow \mathsf{EncCt}(x)$ and then outputs a vector of corruption-induced ciphertext polynomials $(\check{c}_M, \mathbf{c}_{\mathsf{cor}}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}_{\mathsf{hon}}), \mathbf{c}'_{\mathsf{cor}}(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}_{\mathsf{hon}}, \boldsymbol{\beta}_{\mathsf{hon}}), \mathbf{c}''_{\mathsf{cor}}(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta}_{\mathsf{hon}}))$, where

$$\check{c}_{\mathsf{cor},i} = \sum_{\substack{j \in \overline{[w_1]} \\ k \in [n_b] \setminus \mathfrak{b}}} \eta_{1,i,j,k} s_j b_k \; + \sum_{j \in [w_2]} \hat{\eta}_{1,i,j} \hat{s}_j \; + \sum_{\substack{j \in \overline{[w_1]} \\ k \in \mathfrak{b}}} \check{\eta}_{1,i,j,k} s_j \; ,$$

where $\check{\eta}_{1,i,j,k} = \eta_{1,i,j,k} \mathsf{Ev}_b(k)$,

$$\check{c}'_{\mathsf{cor},i} = \sum_{\substack{j \in \mathfrak{a}_1 \\ j' \in \overline{[w_1]}}} \check{\eta}'_{1,i,j,j'} s_{j'} \; + \sum_{\substack{j \in [n_\alpha] \setminus \mathfrak{a}_1 \\ j' \in \overline{[w_1]}}} \eta'_{1,i,j,j'} \alpha_j s_{j'}$$

$$+ \sum_{\substack{j \in \mathfrak{a}_2 \\ j' \in \overline{[w_1]}}} \check{\eta}'_{2,i,j,j'} s_{j'} \; + \sum_{\substack{j \in [n_\beta] \setminus \mathfrak{a}_2 \\ j' \in \overline{[w_1]}}} \eta'_{2,i,j,j'} \beta_j s_{j'} \; + \sum_{j \in [w'_2]} \hat{\eta}'_{i,j} \tilde{s}_j \; ,$$

where $\check{\eta}'_{1,i,j,j'} = \eta'_{1,i,j,j'} \mathsf{Ev}_\alpha(j)$ and $\check{\eta}'_{2,i,j,j'} = \eta'_{2,i,j,j'} \mathsf{Ev}_\beta(j)$,

$$\check{c}''_{\mathsf{cor},i} = \sum_{\substack{j \in \overline{[w_1]} \\ k \in \mathfrak{a}_2}} \check{\eta}_{2,i,j,k} s_j \; + \sum_{\substack{j \in \overline{[w_1]} \\ k \in [n_\beta] \setminus \mathfrak{a}_2}} \eta_{2,i,j,k} s_j \beta_k \; + \sum_{j \in [w'_2]} \hat{\eta}_{2,i,j} \tilde{s}_j \; ,$$

where $\check{\eta}_{2,i,j,k} = \eta_{2,i,j,k} \mathsf{Ev}_\beta(k)$, and

$$\check{c}_M = \sum_{j \in [w'_2]} \zeta_j \tilde{s}_j \; + \sum_{\substack{j \in \mathfrak{a}_1 \\ j' \in \overline{[w_1]}}} \check{\zeta}_{1,j,j'} s_{j'} \; + \sum_{\substack{j \in [n_\alpha] \setminus \mathfrak{a}_1 \\ j' \in \overline{[w_1]}}} \zeta_{1,j,j'} \alpha_j s_{j'}$$

$$+ \sum_{\substack{j \in \mathfrak{a}_2 \\ j' \in \overline{[w_1]}}} \check{\zeta}_{2,j,j'} s_{j'} \; + \sum_{\substack{j \in [n_\beta] \setminus \mathfrak{a}_2 \\ j' \in \overline{[w_1]}}} \zeta_{2,j,j'} \beta_j s_{j'} \; ,$$

where $\check{\zeta}_{1,j,j'} = \zeta_{1,j,j'} \mathsf{Ev}_\alpha(j)$ and $\check{\zeta}_{2,j,j'} = \zeta_{2,j,j'} \mathsf{Ev}_\beta(j)$.

Remark 3. To simplify the specification of the inputs to the corruption-induced key and ciphertext encoding algorithms, we use the fact that the original key and ciphertext encodings are deterministic.

We adjust our security definitions (for trivial and strong MK-MC security) to take into account the corruption-induced encodings. To ensure secure against any integer that is sampled for the corrupted variables, we define the maps $\mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b$, which map each corruptable variable to a value in $\mathbb{Z}_p$.

Definition 11 (Trivial security under corruptions for PES-ISA)**.** Let $(n_\alpha, n_b, n_\beta, \boldsymbol{\alpha}, \mathbf{b}, \boldsymbol{\beta}) \leftarrow \mathsf{Param}(\mathsf{par})$ and $\mathfrak{a}_1 \subsetneq [n_\alpha]$, $\mathfrak{a}_2 \subsetneq [n_\beta]$, $\mathfrak{b} \subsetneq [n_b]$ be indices of corruptable variables with P_{cor} being the predicate that expands the predicate P with respect to the corruptable variables. We call a PES-ISA for predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ trivially secure under corruption of $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$ if, for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ with $P_{\mathsf{cor}}(x, y) = 0$, and corruption-induced polynomials $\check{c}_M, \mathbf{c}_{\mathsf{cor}}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}_{\mathsf{hon}}), \mathbf{c}'_{\mathsf{cor}}(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}_{\mathsf{hon}}, \boldsymbol{\beta}_{\mathsf{hon}}), \mathbf{c}''_{\mathsf{cor}}(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta}_{\mathsf{hon}})$, $\mathbf{k}_{\mathsf{cor}}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}_{\mathsf{hon}}, \boldsymbol{\beta}_{\mathsf{hon}}, \mathbf{b}_{\mathsf{hon}})$, where $\mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b$ map each corruptable variable to a value in $\mathbb{Z}_p$, it holds that

$$\mathbf{e}' \mathbf{c}'^{\mathsf{T}}_{\mathsf{cor}} + \mathbf{e}'' \mathbf{c}''^{\mathsf{T}}_{\mathsf{cor}} + \mathbf{s}\mathbf{E}\mathbf{k}^{\mathsf{T}}_{\mathsf{cor}} + \mathbf{c}_{\mathsf{cor}} \overline{\mathbf{E}} \mathbf{r}^{\mathsf{T}} \neq \check{c}_M,$$

for all $(\mathbf{e}', \mathbf{e}'', \mathbf{E}, \overline{\mathbf{E}}) \in (\mathbb{Z}_p^{w_4}, \mathbb{Z}_p^{w_5}, \mathbb{Z}_p^{(w_1+1)\times m_3}, \mathbb{Z}_p^{w_3\times m_1})$. This is equivalent to stating that the span consisting of $\mathbf{c}'_{\mathsf{cor}}$, $\mathbf{c}''_{\mathsf{cor}}$, all combinations of $\mathbf{s}$ and $\mathbf{k}_{\mathsf{cor}}$ and all combinations of $\mathbf{c}_{\mathsf{cor}}$ and $\mathbf{r}$ does not contain $\check{c}_M$.

Definition 12 (Strong MK-MC security under corruptions)**.** We call a PES-ISA for predicate $P\colon \mathcal{X}\times\mathcal{Y}\to\{0,1\}$ strongly multi-key and multi-ciphertext secure under corruptions, with corruptable variables $\mathfrak{a}_1 \subsetneq [n_\alpha]$, $\mathfrak{a}_2 \subsetneq [n_\beta]$, $\mathfrak{b} \subsetneq [n_b]$ and the predicate P_{cor} that expands the predicate P with respect to the corruptable variables, if it satisfies the strong MK-MC security property (Definition 7) for the corruption-induced key and ciphertext encodings. More formally, we require that, for all $n_k, n_c = \mathsf{poly}(\lambda)$, $x_1, ..., x_{n_c} \in \mathcal{X}$ and $y_1, ..., y_{n_k} \in \mathcal{Y}$ with $P_{\mathsf{cor}}(x_i, y_j) = 0$ and

$$\begin{gathered}\mathbf{c}_{x_1,\dots,x_{n_c}},\quad \mathbf{c}'_{x_1,\dots,x_{n_c}},\quad \mathbf{c}''_{x_1,\dots,x_{n_c}},\quad \mathbf{s}_{x_1,\dots,x_{n_c}},\\ \mathbf{c}_{\mathbf{M}} = (c_{M,x_1}, ..., c_{M,x_{n_c}}),\quad \mathbf{k}_{y_1,\dots,y_{n_k}},\quad \mathbf{r}_{y_1,\dots,y_{n_k}},\end{gathered}$$

which are obtained by running, for all i and j, the corruption-induced ciphertext encodings $\mathrm{CorruptCt}(x_i, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}, \mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b)$, and corruption-induced key encodings $\mathrm{CorruptKey}(y_j, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}, \mathsf{Ev}_\alpha, \mathsf{Ev}_\beta, \mathsf{Ev}_b)$, where $\mathsf{Ev}_\alpha\colon \mathfrak{a}_1 \to \mathbb{Z}_p$, $\mathsf{Ev}_\beta\colon \mathfrak{a}_2 \to \mathbb{Z}_p$ and $\mathsf{Ev}_b\colon \mathfrak{b} \to \mathbb{Z}_p$ be evaluation functions that map all corruptable variables to integers, and then concatenating the resulting encodings, we have

$$\mathbf{e}\cdot\mathbf{p}_{\mathsf{all}}{}^{\intercal} \neq \mathbf{e}_{\mathbf{M}}\mathbf{c}_{\mathbf{M}}{}^{\intercal},$$

for all $\mathbf{e} \in \mathbb{Z}_p^{|\mathbf{p}_{\mathsf{all}}|}$, where

$$\mathbf{p}_{\mathsf{all}} = \boldsymbol{\alpha}_{\mathsf{hon}} \| \mathbf{c}'_{x_{1,\dots,n_c}} \| (\mathbf{p}_{\mathsf{combis}} \setminus \mathbf{p}_{\mathsf{special}}),$$

where

$$\mathbf{p}_{\mathsf{combis}} = \{p_i p_j \mid p_i, p_j \in (1, \boldsymbol{\beta}_{\mathsf{hon}}, \mathbf{b}_{\mathsf{hon}}, \mathbf{s}_{x_{1,\dots,n_c}}, \mathbf{k}_{y_{1,\dots,n_k}}, \mathbf{c}_{x_{1,\dots,n_c}}, \mathbf{c}''_{x_{1,\dots,n_c}}, \mathbf{r}_{y_{1,\dots,n_k}})\}$$

consists of all combinations that can be made and

$$\mathbf{p}_{\mathsf{special}} = \{s_j\beta_k \mid \forall i \in [n_c], s_j \in \mathbf{s}_{x_i}, \beta_k \in \boldsymbol{\beta}_{\mathsf{hon}}\ [\ \exists i' \in [m_3]\ [\ \eta_{2,i',j,k} \neq 0 \vee \eta'_{2,i',j,k} \neq 0\ ]\ ]\ \}$$

consists of all combinations $s_j\beta_k$ that do not occur in the ciphertext polynomials.

To prove security against corruptions, we use the results from Section 3.3, and combine it with the insight that all the monomials in the entries in $\mathbf{v}$ for which $\mathbf{w}$ is 0 are "irrelevant" to the attacker's abilities. Roughly, if $\mathbf{w}$ is 0 in all entries associated with a corrupted variable, they do not help the attacker break the scheme, as adding a monomial p_i that contains the corrupted variable to the set of possible combinations will not affect the security. In particular, because the vector $\mathbf{w}$ is 0 in the associated entry, it will also be orthogonal to $\mathbf{p}_i$, where $p_i = \mathbf{p}_i \cdot \mathbf{v}^{\intercal}$, and therefore, it will still be in the kernel of the matrix with $\mathbf{p}_i$ appended to it in the last row. Therefore, we define our kernel property for MK-MC security under corruptions as follows.

Definition 13 (Kernel property for MK-MC under corruptions)**.** Consider a PES-ISA that satisfies the kernel property for MK-MC security (Definition 9). Furthermore, let $\mathfrak{a}_1 \subsetneq [n_\alpha]$, $\mathfrak{a}_2 \subsetneq [n_\beta]$, $\mathfrak{b} \subsetneq [n_b]$ and the predicate P_{cor} that expands the predicate P with respect to the variables $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$ be such that, for all $x \in \mathcal{X}, y \in \mathcal{Y}$ with $P_{\mathsf{cor}}(x, y) = 0$, the vector $\mathbf{w}_{x,y}$ has the following property. If $(\mathbf{v}_{x,y})_i$ is a monomial that is a product of at least one of the variables, i.e., α_j with $j \in \mathfrak{a}_1$, β_j with $j \in \mathfrak{a}_2$ or b_k with $k \in \mathfrak{b}$, then $(\mathbf{w}_{x,y})_i = 0$. We say that the PES-ISA satisfies MK-MC security under corruptions, with corruptable variables $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$.

We formulate the following theorem that we apply in our security proof in the generic group model in Theorem 4.

Theorem 1. *Consider a PES-ISA with the kernel property for MK-MC security under corruptions. Then it is strongly multi-key and multi-ciphertext secure under corruptions.*

Proof. Let $\mathbf{p_{enc}}_{x,y} = \mathbf{M}_{x,y} \cdot \mathbf{v}_{x,y}^{\intercal}$ denote the matrix decomposition for the key-ciphertext combinations *before* corruption. Because the PES-ISA satisfies the kernel property for MK-MC security under corruptions, we know that, for each x and y, there is a vector $\mathbf{w}_{x,y}$ in the kernel of $\mathbf{M}_{x,y}$ such that $\mathbf{tv}_{x,y} \cdot \mathbf{w}_{x,y}^{\intercal} \neq 0$ and, for each entry i associated with a corruptable variable (in $\mathbf{v}_{x,y}$), we have that $(\mathbf{w}_{x,y})_i = 0$. Now, we can use $\mathbf{w}_{x,y}$ to construct a vector $\mathbf{w}_{x,y,\mathrm{cor}}$ for the corruption-induced encodings. First, let $\mathbf{p_{enc}}_{x,y,\mathrm{cor}}$ denote the combinations of keys and ciphertexts (as in Definition 8) of the corruption-induced key and ciphertext encodings. We can construct a matrix decomposition $\mathbf{p_{enc}}_{x,y,\mathrm{cor}} = \mathbf{M}_{x,y,\mathrm{cor}} \cdot \mathbf{v}_{x,y,\mathrm{cor}}^{\intercal}$ from $\mathbf{M}_{x,y}$ and $\mathbf{v}_{x,y}$ as follows. First, we split $\mathbf{M}_{x,y}$ and $\mathbf{v}_{x,y}$, without loss of generality, in the part associated with honestly generated variables and corruptable variables, respectively:

$$\mathbf{M}_{x,y} = \begin{pmatrix} \mathbf{M}_{x,y,1} & \mathbf{M}_{x,y,2} \end{pmatrix}, \qquad \mathbf{v}_{x,y} = \begin{pmatrix} \mathbf{v}_{x,y,1}^{\intercal} \\ \mathbf{v}_{x,y,2}^{\intercal} \end{pmatrix},$$

so that $\mathbf{v}_{x,y,1}$ consists of non-corruptable variables only and each entry in $\mathbf{v}_{x,y,2}$ consists of at least one corruptable variable. Note that the associated kernel vector $\mathbf{w}_{x,y} = \mathbf{w}_{x,y,1} \| \mathbf{w}_{x,y,2}$ is such that $\mathbf{w}_{x,y,2}$ is all-zero and $\mathbf{tv}_{x,y} \cdot \mathbf{w}_{x,y}^{\intercal} = \mathbf{tv}_{x,y,1} \cdot \mathbf{w}_{x,y,1}^{\intercal} \neq 0$ (where $\mathbf{tv}_{x,y} = \mathbf{tv}_{x,y,1} \| \mathbf{tv}_{x,y,2}$). We now construct a decomposition $\mathbf{M}_{x,y,2,\mathrm{cor}} \cdot \mathbf{v}_{x,y,2,\mathrm{cor}}^{\intercal}$ from $\mathbf{M}_{x,y,2} \cdot \mathbf{v}_{x,y,2}^{\intercal}$ by treating the corruptable variables as integers (to be moved to the matrix) and the honestly generated variables as regular variables. In this way, we obtain the matrix decomposition

$$\mathbf{p_{enc}}_{x,y,\mathrm{cor}} = \begin{pmatrix} \mathbf{M}_{x,y,1} & \mathbf{M}_{x,y,2,\mathrm{cor}} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_{x,y,1}^{\intercal} \\ \mathbf{v}_{x,y,2,\mathrm{cor}}^{\intercal} \end{pmatrix},$$

where each entry in $\mathbf{v}_{x,y,2,\mathrm{cor}}$ is of the form 1 or $r_j s_{j'}$. (This follows from our definition of corruption-induced encodings.) We similarly construct $\mathbf{tv}_{x,y,\mathrm{cor}} = \mathbf{tv}_{x,y,1} \| \mathbf{tv}_{x,y,2,\mathrm{cor}}$. Finally, we construct $\mathbf{w}_{x,y,\mathrm{cor}} = \mathbf{w}_{x,y,1} \| \mathbf{0}^{|\mathbf{v}_{x,y,2,\mathrm{cor}}|}$, which is in the kernel of $\mathbf{M}_{x,y,\mathrm{cor}}$ and we have $\mathbf{tv}_{x,y,\mathrm{cor}} \cdot \mathbf{w}_{x,y,\mathrm{cor}}^{\intercal} = \mathbf{tv}_{x,y,1} \cdot \mathbf{w}_{x,y,1}^{\intercal} \neq 0$. Via the MK-MC security results (without corruption), this proves that the PES-ISA is MK-MC secure.

Then, to achieve strong MK-MC security, we show that this property is complete in a similar sense as in Proposition 3. This means that we want to prove that the occurrences of $r_j s_{j'}$ combinations in the critical key-ciphertext combinations that are introduced with these corruptions do not help the attacker break the scheme. To show this, we consider a set of polynomials $\{p_1, ..., p_n\}$ such that each p_i may contain monomials that also occur in $\mathbf{v}_{x,y,2,\mathrm{cor}}$ (for any x, y). (In particular, we had already ruled out that p_i can contain monomials in $\mathbf{v}_{x,y,1}$ in the proof of Lemma 3.) Hence, the vector $\mathbf{v}_{p_i}$ in the decomposition $p_i = \mathbf{p}_i \cdot \mathbf{v}_{p_i}^{\intercal}$ is not disjoint from $\mathbf{v}_{x,y,\mathrm{cor}}$. Without loss of generality, we can split $\mathbf{v}_{x,y,\mathrm{cor}} = \mathbf{v}'_{x,y,1} \| \mathbf{v}'_{x,y,2}$ and $\mathbf{v}_{p_i} = \mathbf{v}_{p_i,1} \| \mathbf{v}_{p_i,2}$, such that $\mathbf{v}'_{x,y,1} = \mathbf{v}_{p_i,1}$ and $\mathbf{v}'_{x,y,2}$ and $\mathbf{v}_{p_i,2}$ are disjoint. (We split $\mathbf{M}_{x,y,\mathrm{cor}}$ and $\mathbf{p}_i$ accordingly.) Note that $\mathbf{v}'_{x,y,1}$ contains entries that also occur in $\mathbf{v}_{x,y,2,\mathrm{cor}}$ but not in $\mathbf{v}_{x,y,1}$. We can now construct a vector $\mathbf{w}'$ from $\mathbf{w}_{x,y,\mathrm{cor}}$, which is orthogonal to the matrix

$$\begin{pmatrix} \mathbf{M}'_{x,y,\mathrm{cor},1} & \mathbf{M}'_{x,y,\mathrm{cor},2} & \mathbf{0} \\ \mathbf{p}_{i,1} & \mathbf{0} & \mathbf{p}_{i,2} \end{pmatrix},$$

by setting $\mathbf{w}' = \mathbf{w}'_{x,y,1} \| \mathbf{w}'_{x,y,2} \| \mathbf{0}^{|\mathbf{v}_{p_i,2}|}$, where $\mathbf{w}'_{x,y,1} = \mathbf{0}^{|\mathbf{v}'_{x,y,1}|}$ because of what we have proven earlier. Thus, similarly as in the proof of Proposition 3, $\{p_1, ..., p_n\}$ do not affect the MK-MC security of the PES-ISA. □

3.5 Equivalence with SSSP

We show that the special selective symbolic property as formulated in Definition 4 and the kernel property for MK-MC security under corruptions (Definition 13) are equivalent. Our proof for this is constructive, and can therefore be used to generated proofs for SSSP.

Theorem 2. *The following two statements are equivalent.*

- *The PES-ISA satisfies the special selective symbolic property with corruptable variables* $\mathfrak{a}_1 \subsetneq [n_\alpha]$, $\mathfrak{a}_2 \subsetneq [n_\beta]$, $\mathfrak{b} \subsetneq [n_b]$ *and the predicate* P_{cor} *that expands the predicate* P *with respect to the corruptable variables.*
- *The PES-ISA satisfies the kernel property for MK-MC security under corruptions, with corruptable variables* $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$.

Proof. Proving that the kernel property holds when we have a proof for SSSP is almost trivial. In particular, we can construct a vector $\mathbf{w}_{x,y}$ by taking an SSSP proof and constructing its associated $\mathbf{p_{enc}}_{x,y}$ vector and matrix decomposition $\mathbf{M}_{x,y} \cdot \mathbf{v}_{x,y}^\intercal$. For each entry in $\mathbf{v}_{x,y}$, we substitute the vectors and matrices from the SSSP proof. Because the SSSP proof requires that substituting the key and ciphertext polynomials yields all-zero vectors, this yields a kernel vector. Because the proof requires that substituting c_M with the appropriate vectors yields a nonzero value, the resulting vector is not orthogonal to the target vector. Furthermore, because of the requirements that corrupted variables are substituted with all-zero vectors and matrices, all entries in $\mathbf{w}_{x,y}$ that are associated with corrupted variables are 0. Lastly, the part of $\mathbf{w}_{x,y}$ that is associated with the entries $\alpha_j s_{j'}$ and $\tilde{s}_j$ is the same for each y, because these are generated with only x as input.

Proving that SSSP holds when we have a proof for the kernel property is more intricate. In particular, we need to prove that the vector $\mathbf{w}_{x,y}$ can be decomposed. For this part of the proof, we use a similar approach as in the proof of Theorem 3 in [dlPVA23]. We first note that the entries of $\mathbf{w}_{x,y}$ that are associated with $\tilde{s}_j$ directly describe the substitution values for the lone ciphertext variables $\tilde{s}_j$. For the substitution vectors for s_j, we assign $\mathbf{1}_{j+1}^{d_1}$, where $d_1 = w_1 + 1$ (i.e., the number of non-lone key variables). The rough idea behind this is that this vector selects the $(j+1)$-th row of the substitution matrices for the common variables, and the $(j+1)$-th entry of the substitution vectors for the master-key and semi-common variables. Consequently, we encode the substitution vectors for the master-key and semi-common variables with the entries in $\mathbf{w}_{x,y}$ that are associated with $\alpha_j s_{j'}$ and $\beta_j s_{j'}$. That is, the $(j'+1)$-th entry of these vectors is the associated value in $\mathbf{w}_{x,y}$. Similarly, we can encode the substitution vectors for the lone key variables $\hat{r}_j$ by taking the values in the entries of $\mathbf{w}_{x,y}$ that are associated with $\hat{r}_j s_{j'}$ and setting the $(j'+1)$-th entry of the substitution to be that value.

To encode the substitution matrices for the common variables (as well as the substitutions for the lone ciphertext variables $\hat{s}_j$), we need a more advanced approach. For the corruptable common variables, we simply output all-zero matrices. For the non-corruptable variables, we take the following steps. We first consider the matrix decomposition for the ciphertext encodings $\mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}) = \mathbf{M}_c \cdot \mathbf{v}_c^\intercal$. Let $\mathcal{V}_c = \{\mathsf{v}_{c,1}, ..., \mathsf{v}_{c,d_2}\}$ denote a basis for the kernel of $\mathbf{M}_c$, and note that d_2 (as in the definition of SSSP) is determined by the size of this kernel. The entries of $\mathbf{v}_c$ are of the form $s_j b_k$ and $\hat{s}_j$, meaning that each kernel vector in $\mathcal{V}_c$ has entries associated with $\hat{s}_j$. We construct the substitution vectors for $\hat{s}_j$ by taking the associated entry in the i-th vector $\mathsf{v}_{c,i}$ in $\mathcal{V}_c$ and setting the i-th entry of the substitution vector to be that value. For the substitution matrices for the common variables b_k, we extend this process to the two-dimensional setting. The value that we place in the $(j+1)$-th row and i-th column of the substitution matrix for b_k is the value in the entry associated with $s_j b_k$ of $\mathsf{v}_{c,i}$. Hence, the $(j+1)$-th row encodes the basis for the kernel of $\mathbf{M}_c$ associated with s_j and b_k. Note that this process already ensures that substituting the ciphertext polynomials in $\mathbf{c}$ yields all-zero vectors.

To encode the substitution vectors for the non-lone key variables r_j, we use that the kernel bases for $\mathbf{M}_{x,y}$ and $\mathbf{M}_c$ are closely related. In particular, the rows of $\mathbf{M}_{x,y}$ that are associated with the key-ciphertext combinations $r_j c_i$ are just "expanded" versions of the rows in $\mathbf{M}_c$ associated with c_i. Essentially, $\mathbf{M}_c$ is a submatrix of $\mathbf{M}_{x,y}$. As is proven in [dlPVA23], this means that the kernel of $\mathbf{M}_{x,y}$ is a subspace of the kernel of the submatrix of $\mathbf{M}_{x,y}$ implied by $\mathbf{M}_c$, meaning that $\mathbf{w}_{x,y}$ can be written as a linear combination of expanded versions of $\mathsf{v}_{c,1}, ..., \mathsf{v}_{c,d_2}$ (for each r_j). Conversely, we can, for each r_j, "cut out the $\mathbf{c}$-part" from $\mathbf{w}_{x,y}$ and write each of these vectors as a linear combination of the basis vectors $\mathsf{v}_{c,1}, ..., \mathsf{v}_{c,d_2}$. The coefficients of this linear combination are exactly the entries of the substitution vector for r_j, where the coefficient for $\mathsf{v}_{c,i}$ is the i-th entry of the substitution vector.

Note that this process yields substitution vectors and matrices for the variables such that the requirements for SSSP hold. This follows from the same insights as for the other direction of this proof. □

3.6 Computer-assisted proofs with ACABELLA

One of the main advantages of SSSP is that it has been shown, for the more restricted class PES-AC17, that it can be proven with computer-assisted tools such as ACABELLA [dlPVA23]. To benefit from this advantage, we extend the ACABELLA tool to compute substitution vectors and matrices that satisfy the SSSP requirements for PES-ISA (and, by extension, also PES-Ven23), if such vectors and matrices exist. The code for the extension is available at https://github.com/lincolncryptools/ISABELLA.

High-level description of our extension. The tool takes as input a JSON file with a description of the PES-ISA. To generate the required substitution vectors and matrices, our tool extension performs the following steps:

- It first parses the inputs and verifies the correctness, i.e., whether the input encodings satisfy the scheme format in Definition 3;
- It then computes the kernel vector $\mathbf{w}$ that functions as a proof for the kernel property for MK-MC (with or without corruptions is determined by whether the input describes corruptable variables);
- It decomposes the kernel vector $\mathbf{w}$ into proofs.

The input file and parser. Compared to the original tool, our JSON input takes one extra input: the corruptable variables. Furthermore, much like ACABELLA, we have minimized the required user input. First, the tool does not distinguish between $\mathbf{c}'$ and $\mathbf{c}''$, because the substitution vectors for the master-key and semi-common variables are generated in the same way. Second, the user can give all the public-key variables (master-key, semi-common, common) in one entry field. The same holds for all the key encodings (which includes non-lone key variables and key polynomials) and all the ciphertext encodings (which includes the non-lone ciphertext variables and all the ciphertext polynomials). The tool derives their role in the PES from this user input. We describe the algorithm for the parser and correctness checker in Appendix A.1.

Finding a suitable kernel vector. Compared to the proving functionality for PES-AC17, we have several hurdles to overcome when generating a suitable kernel vector that proves the MK-MC security property (Definitions 9 and 13). In particular, the requirements for the kernel vector are, in some sense, a bit more abstract than for PES-AC17. Furthermore, because we allow corruptions, the requirements are also stronger, i.e., certain entries need

to be 0. To enable the latter, we adapt the algorithms that ACABELLA provides to transform a basis into a basis for which the entries for some specified set are set to zero. The more difficult requirement to overcome is that the part of the vector $\mathbf{w}$ that is associated with $\alpha_j s_{j'}, \beta_j s_{j'}$ or $\tilde{s}_{j'}$ needs to be the same for each key and ciphertext predicate. In contrast, for PES-AC17, this part of the vector needs to be $(1, 0, ..., 0)$. This property is too strong for PES-ISA. (In fact, none of the decentralized schemes in the appendix or in [Ven23] have this property.) Additionally, because ACABELLA cannot interpret predicates, nor gets it a second input for the keys, solving this problem is essentially impossible. To circumvent the problem, our tool extension computes a basis for candidates for the vector $\mathbf{w}$ that are independent of the keys, instead. The motivation for this is that the substitution vectors for $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are generated independently of the key inputs y. Hence, any vector $\mathbf{w}$ for which the stronger property holds is also key independent and must therefore be in this basis. For future work, if ACABELLA is updated with a parser that does interpret the key and ciphertext predicates, our tool extension can be further extended to compute a better candidate. Currently, it outputs any kernel vector that is not orthogonal to the target vector.

Computing a basis for key-independent kernel vectors. The main technical contribution that our tool extension provides compared to ACABELLA is our algorithm to compute a basis for suitable kernel candidates for $\mathbf{w}$. As a starting point, we first compute a basis for $\mathbf{M}$, where $\mathbf{M}$ is as in the matrix decomposition of the vector of key-ciphertext combinations, $\mathbf{p_{enc}}_{x,y} = \mathbf{M}_{x,y} \cdot \mathbf{v}^{\intercal}_{x,y}$. (Note that we call it $\mathbf{M}$ because the x and y are implicit to the tool.) Then we apply an algorithm to transform the basis into a basis for the kernel of $\mathbf{M}$ whose vectors set the entries associated with corruptable variables to 0. Using this basis as input, we transform it into a basis that additionally ensures that the entries associated with $\alpha_j s_{j'}, \beta_j s_{j'}$ or $\tilde{s}_{j'}$ are key independent. We define a vector to be key independent if none of the symbols in the entries occur in the keys but not in the ciphertext. For example, if x_{att} is a variable that occurs in the keys but not the ciphertexts, and it occurs in one of the entries, then the vector is key dependent. To transform the basis, we leverage the following givens: the ciphertext encodings $\mathbf{c}'$ are key independent by definition, and its matrix decomposition has $\alpha_j s_{j'}, \beta_j s_{j'}$ and $\tilde{s}_{j'}$ in the entries of the vector. Furthermore, from some linear algebra arguments, it follows that the basis vectors of the kernel truncated to the entries associated with $\alpha_j s_{j'}, \beta_j s_{j'}$ or $\tilde{s}_{j'}$ is in the kernel of the matrix of this decomposition. This means that we can write each truncated vector as a linear combination of basis vectors for the kernel of $\mathbf{c}'$, which are key independent by definition. The goal is then to find linear combinations of the basis vectors that are key independent, linearly independent and whose span is "complete" in that it describes all key independent vectors in the kernel. To this end, we leverage the algebraic properties that the reduced row echelon form (RREF) of a matrix provides us. Specifically, we put the coefficients of these linear combinations in the matrix (i.e., each row corresponds to a basis vector of the kernel of the key-ciphertext vector) and deduce the RREF. The nonzero rows are linearly independent and span the whole kernel. The rough idea is then to remove all the key dependent rows and let the remaining nonzero rows determine the basis of key-independent kernel vectors.

Proofs of correctness and completeness of the algorithm. We have proven that our algorithms are correct and complete, i.e., the steps yield a basis for the kernel of the matrix for the key-ciphertext combinations, with the additional restriction that the vectors are key independent and 0 in the corruptable-variable entries. These proofs can be found in Appendix B.

Decomposing the suitable kernel vector. Once we have computed a suitable candidate for $\mathbf{w}$, we decompose it into a proof for SSSP, using the proof for Theorem 2. This algorithm is fairly similar to that of ACABELLA for the PES-AC17 class of schemes.

Full descriptions of the algorithms and proofs of correctness. The full descriptions of the algorithms can be found in Appendix A. The linear-algebra tools that lie at the core of our algorithms (and subsequently provide proofs of correctness for these algorithms) can be found in Appendix B.

4 Our generic compiler

Equipped with our "security core", consisting of PES-ISA and SSSP, we can discuss our compiler and the missing "ingredients" that we need to support advanced functionalities. Roughly, when instantiating our class of PES in the groups, the master public key, secret keys and ciphertexts have the following form:

$$\mathsf{MPK} = \left(e(g,h)^{\boldsymbol{\alpha}}, (g')^{\mathbf{b}}, (g')^{\boldsymbol{\beta}}\right), \qquad \mathsf{SK}_y = \left((g')^{\mathbf{r}}, (g')^{\mathbf{k}(\mathbf{r},\hat{\mathbf{r}},\boldsymbol{\alpha},\boldsymbol{\beta},\mathbf{b})}\right),$$
$$\mathsf{CT}_x = \left(M \cdot e(g,h)^{c_M}, (g')^{\mathbf{s}}, (g')^{\mathbf{c}(\mathbf{s},\hat{\mathbf{s}},\mathbf{b})}, e(g,h)^{\mathbf{c}'(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\alpha},\boldsymbol{\beta})}, (g')^{\mathbf{c}''(\mathbf{s},\tilde{\mathbf{s}},\boldsymbol{\beta})}\right).$$

To support additional functionalities, we use the Ven23 approach to include mappings that assign groups to the above variables (indicating in which groups they will be instantiated), and that indicate that certain variables can be generated via full-domain hashes. We review how this is done in the following subsection. After that, we define the notion of split-predicate mold to allow for a more adaptive interaction between the user (requesting a key for y) and the authority (generating the key), before we turn to the final ISABELLA compiler in Section 4.3.

4.1 Distribution of encodings and FDHs

In order to instantiate the PES-ISA in a pairing group, we need to specify which components are instantiated in which of the groups. We also want to allow the implicit generation of variables via full-domain hash functions. In both cases we need to ensure that correctness is preserved. We model this via distributions, similar to the Ven23 compiler.

Distribution of the encodings. Our compiler takes as input explicit definitions for the distribution of the encodings over the two source groups $\mathbb{G}$ and $\mathbb{H}$ when they are instantiated. Such a distribution should ensure that the correctness of the PES is preserved, such that the correctness of the ABE scheme is also guaranteed. In particular, for the correctness of the decryption algorithm, we require that each pair of key and ciphertext encodings that needs to be paired has one input in $\mathbb{G}$ and one in $\mathbb{H}$. Furthermore, to ensure that encryption can be performed correctly, the master public keys required in computing a ciphertext encoding element need to be in the same group. We additionally also have some restrictions on the distributions of $\mathbf{s}$ and $\boldsymbol{\beta}$. In particular, these need to be in the same group when they occur in a product in $\mathbf{c}''$. Note that this last requirement also ensures that $\mathbf{p}_{\mathsf{special}}$ can be excluded from all products that can be generated by the attacker in Definitions 7 and 12.

Definition 14 (Distribution of the encodings over $\mathbb{G}$ and $\mathbb{H}$)**.** Let $\Gamma_{\text{PES-ISA}}$ = (Param, EncKey, EncCt, Pair) be a PES-ISA for a predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and let $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}_T$ be three groups. Let $\mathcal{E} = (\boldsymbol{\beta}, \mathbf{b}, \mathbf{r}, \mathbf{k}, \mathbf{s}, \mathbf{c}', \mathbf{c}'')$ denote the set of possible encodings and non-lone variables that can be sampled with Param, EncKey and EncCt, and let $\mathcal{E}' \subseteq \mathcal{E}$ denote its subset containing the master key variables $\boldsymbol{\alpha}$ and ciphertext encodings $\mathbf{c}'$. Then, we define $\mathfrak{D}\colon \mathcal{E} \to \{\mathbb{G}, \mathbb{H}\}$ to be the distribution of Γ over $\mathbb{G}$ and $\mathbb{H}$ such that

the correctness of the encoding is preserved. This is the case, if for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ such that $P(x, y) = 1$, it holds that

- for all $i \in [m_3]$, $j \in \overline{[w_1]}$, if $\mathfrak{D}(k_i) = \mathfrak{D}(s_j)$, then $\mathbf{E}_{j,i} = 0$;
- for all $i \in [w_3]$, $j \in [m_1]$, if $\mathfrak{D}(c_i) = \mathfrak{D}(r_j)$, then $\overline{\mathbf{E}}_{i,j} = 0$;
- for all $k \in [n_b]$ for which there exist some $i \in [w_3], j \in \overline{[w_1]}$ with $\eta_{i,j,k} \neq 0$, we have $\mathfrak{D}(b_k) = \mathfrak{D}(c_i)$;
- for all $k \in [n_\beta]$ for which there exist some $i \in [w_5], j \in \overline{[w_1]}$ with $\eta_{2,i,j,k} \neq 0$, we have $\mathfrak{D}(\beta_k) = \mathfrak{D}(c''_i)$;
- for all $j \in \overline{[w_1]}, k \in [n_\beta]$ for which there exists some $i \in [w_5]$ with $\eta_{2,i,j,k} \neq 0$, we have $\mathfrak{D}(s_j) = \mathfrak{D}(\beta_k)$.

Full-domain hashes and random oracles. When variables are generated implicitly by a full-domain hash (FDH), the description of the scheme needs to specify the hash input strings. Our definition also allows for explicit domain separation. For the security proofs, we require that an FDH can be modeled as a random oracle.

Definition 15 (FDH-generated encoding variables)**.** Let $\Gamma_{\text{PES-ISA}} = (\text{Param}, \text{EncKey}, \text{EncCt}, \text{Pair})$ be a PES-ISA for a predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$. Let $\mathcal{E} = (\mathbf{b}, \mathbf{r}, \hat{\mathbf{r}}, \mathbf{s}, \hat{\mathbf{s}}, \tilde{\mathbf{s}})$ denote the set of common and (non-)lone key and ciphertext variables that are generated with Param, EncKey and EncCt. Then, we define $\mathcal{F}\colon \mathcal{E} \to \mathbb{N} \times \{0,1\}^*$ to be the mapping that assigns whether the encoding variables are generated by an FDH or not, and if so, what inputs the FDH expect. If not, then the encoding variable is mapped to $(0, \epsilon)$ (where ϵ denotes an empty string). Otherwise, it is mapped to any integer larger than 0. When the FDH is instantiated, it expects the index of the encoding variable as input, e.g., if $\mathcal{F}(b_{\mathsf{att}}) = (2, \mathsf{att})$, then $\mathcal{H}$ expects $(2, \mathsf{att})$ as input in the scheme, and outputs and element in group $\mathfrak{D}(b_{\mathsf{att}})$. For simplicity of notation, when we only need to select the integer output in the first entry of the function $\mathcal{F}$, we denote this as $\mathcal{F}_1$, e.g., $\mathcal{F}_1(b_{\mathsf{att}}) = 2$.

To ensure correctness of the scheme, we place some restrictions on the distributions and full-domain hashes. Like in the Ven23 compiler, we require the distribution over the two source groups to be such that, for any common variable b_k that is provided implicitly by a hash, and each associated encoding k_i and c_i, it holds that they are placed in the same group. Similarly, we can define such a restriction for the other variables. Furthermore, if a non-lone variable and a common variable occur together in a product in one of the polynomials, then it cannot be the case that both are generated by an FDH. We formalize these restrictions as follows.

Definition 16 (Correctness of variables generated by an FDH)**.** Let $\mathfrak{D}$ be as in Definition 14. Then, the following restrictions should hold. For any common variable b_k with $\mathcal{F}_1(b_k) > 0$ (i.e., generated implicitly by the full-domain hash), it holds that:

- For all $i \in [m_3]$, if $\mathfrak{D}(k_i) \neq \mathfrak{D}(b_k)$, then $\delta_{i,j,k} = 0$ for all $j \in [m_1]$;
- For all $i \in [w_3]$, if $\mathfrak{D}(c_i) \neq \mathfrak{D}(b_k)$, then $\eta_{i,j,k} = 0$ for all $j \in \overline{[w_1]}$.

For any non-lone variable r_j or s_j with $\mathcal{F}_1(r_j) > 0$, $\mathcal{F}_1(s_j) > 0$, it holds that:

- For all $i \in [m_3]$, if $\mathfrak{D}(k_i) \neq \mathfrak{D}(r_j)$, then $\delta_{i,j,k} = 0$ for all $k \in [n]$;
- For all $i \in [w_3]$, if $\mathfrak{D}(c_i) \neq \mathfrak{D}(s_j)$, then $\eta_{i,j,k} = 0$ for all $k \in [n]$;
- For all $i \in [m_3], k \in [n]$, if $\delta_{i,j,k} \neq 0$, then $\mathcal{F}_1(b_k) = 0$;

- For all $i \in [w_3], k \in [n]$, if $\eta_{i,j,k} \neq 0$, then $\mathcal{F}_1(b_k) = 0$.

Furthermore, for each $\ell, \ell' \in \mathbb{N}$ with $\ell, \ell' > 0$ and $\ell \neq \ell'$, we require that all the encodings that are mapped to it are of the same kind, e.g., common variables, lone key variables, etc., and that these are mapped to the same group with $\mathfrak{D}$.

Default distributions. There are two distributions that work by default: placing all non-lone variables in $\mathbb{G}$ (or $\mathbb{H}$), all the polynomials $\mathbf{k}, \mathbf{c}$ in $\mathbb{H}$ (or $\mathbb{G}$), and the polynomials $\mathbf{c}''$ in $\mathbb{G}$ (or $\mathbb{H}$). It is easy to see that this satisfies the correctness requirements for the distributions and the FDHs. If no FDHs are used to generate the common variables, it is possible to place all ciphertext components (except $\mathbf{c}'$) in $\mathbb{G}$ (or $\mathbb{H}$) and all key components in $\mathbb{H}$ (or $\mathbb{G}$).

4.2 Split-predicate mold for PES

To generate the master public keys, secret keys and ciphertext more adaptively, we introduce the notion of a split-predicate mold for PES. Roughly, the idea is that the master keys, the secret keys and the ciphertexts can be generated more adaptively if their structure allows this. This is necessary for, e.g., multi-authority ABE, because the keys are generated for the subsets of the whole set possessed by the user that are managed by each authority. Essentially, our definition of split-predicate PES generalizes and formalizes the role of independent encodings of the Ven23 compiler. We take a more general approach so that we can support such adaptivity for all algorithms, and additionally, we allow more generic splits.

Sub-predicate spaces. An important component of our definition for split-predicate mold is the notion of sub-predicate spaces, which splits the predicates into "shares" to be taken as input to the split-predicate encodings. Intuitively, we need to be able to split the key predicate space $\mathcal{Y}$ (and any element therein) in "sub-predicate spaces" such that we can split any key predicate $y \in \mathcal{Y}$ in "sub-predicates". These sub-predicates can be used to reconstruct the original key predicate, and the equivalence of the original predicate is logically preserved. (That is, it should not be the case that the sub-predicates are more powerful and trivially enable us to violate the original predicate.) To check whether the predicate is satisfied for a set of sub-predicates, we define an aggregation function that reconstructs the predicate y from any set of sub-predicates. More formally, we define this as follows. Let $\mathcal{J}$ denote some (potentially exponentially sized) indexing space. Then, let $\{\mathcal{Y}_j\}_{j \in \mathcal{J}'}$ denote any subset of sub-predicate spaces for $\mathcal{Y}$, where $\mathcal{J}' \subseteq \mathcal{J}$ is polynomially sized. Further, we define the aggregation function $\mathsf{Agg}_{\mathrm{key},\mathcal{J}'} \colon \prod_{j \in \mathcal{J}'} \mathcal{Y}_j \to \mathcal{Y}$, such that $\mathsf{Agg}_{\mathrm{key},\mathcal{J}'}(\{y_j\}_{j \in \mathcal{J}'}) = y$. Using these functions, we can efficiently verify whether the ciphertext predicate x is satisfied by a set of key sub-predicates $\{y_j\}_{j \in \mathcal{J}'}$ by aggregating to reconstruct y and evaluating $P(x, y)$.

Tying key sub-predicates to one key predicate. The second problem that we need to address is that we should not be able to aggregate $\{y_j\}_{j \in \mathcal{J}'}$ for different keys. On a syntactic level, we ensure this by introducing an "anchor" aux_y, which is a set of strings that is given as auxiliary input to the split version of the key generation. Roughly, aux_y ties together the sub-predicates $\{y_j\}_{j \in \mathcal{J}'}$ to one split key. Specifically, (aux_y, y_1) and (aux_y, y_2) would be able to aggregate a key to (aux_y, y) but (aux_y, y_1) and (aux'_y, y_2) with $\mathrm{aux}_y \cap \mathrm{aux}'_y = \emptyset$ would not. This trick is similar to how decentralized ABE [LW11] ties the keys generated at different authorities to the same user and can be seen as a generalization of it. In particular, aux_y can be used as input to generate the values associated with the non-lone variables, which can be used to tie different instances of the split key generation to one aux_y.

Definition 17 (Split-predicate mold for PES-ISA)**.** Let $\Gamma_{\text{PES-ISA}} = (\text{Param}, \text{EncKey}, \text{EncCt}, \text{Pair})$ be a PES-ISA for a predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and a prime integer $p \in \mathbb{N}$. We define a split-predicate mold for $\Gamma_{\text{PES-ISA}}$, denoted by $\mathsf{SPM}[\Gamma_{\text{PES-ISA}}]$, as follows. First, we assume that the inputs par can be split in sub-parameters $\{\text{par}_i\}_{i\in\mathcal{I}}$ such that $\mathcal{I}$ denotes a polynomially sized set of strings, and par can be reconstructed from $\{\text{par}_i\}_{i\in\mathcal{I}}$. Second, we assume that the key predicate $y \in \mathcal{Y}$ can be split in "sub-predicates" $\{y_j\}_{j\in\mathcal{J}'}$ where $\mathcal{J}' \subseteq \mathcal{J}$ is some polynomially sized set of strings, and accordingly $\mathcal{Y}$ can be split in sub-predicate spaces $\{\mathcal{Y}_j\}_{j\in\mathcal{J}'}$, such that $y_j \in \mathcal{Y}_j$ and y can be reconstructed from $\{y_j\}_{j\in\mathcal{J}'}$ with some aggregation function $\mathsf{Agg}_{\text{key},\mathcal{J}'}\colon \prod_{j\in\mathcal{J}'} \mathcal{Y}_j \to \mathcal{Y}$, such that $\mathsf{Agg}_{\text{key},\mathcal{J}'}((y_j)_{j\in\mathcal{J}'}) = y$. Additionally, we define a mapping $\mathsf{AssignMPK}\colon \mathcal{J} \to \mathcal{P}(\mathcal{I})$, where $\mathcal{P}(\mathcal{I}) = \{\mathcal{I}' \subseteq \mathcal{I}\}$ is the power set of $\mathcal{I}$, which maps every sub-predicate index $j \in \mathcal{J}$ to a subset of the parameter indices $\mathcal{I}$. Third, we assume that the algorithms Param, EncKey can be split (the EncCt algorithm remains the same):

- $\text{SplitParam}_i(\text{par}_i) \to (\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{b}_i)$: This algorithm is the same as Param in Definition 3, except that it takes as input the parameter index $i \in \mathcal{I}$ and the sub-parameters par_i and outputs a subset of the original algorithm.
- $\text{SplitEncKey}_j(y_j, \text{aux}_y) \to \mathbf{k}_j(\mathbf{r}_j, \hat{\mathbf{r}}_j, \boldsymbol{\alpha}_{\mathsf{AssignMPK}(j)}, \boldsymbol{\beta}_{\mathsf{AssignMPK}(j)}, \mathbf{b}_{\mathsf{AssignMPK}(j)})$: This algorithm is the same as $\text{EncKey}(y)$ in Definition 3, except that it takes as input the sub-predicate index $j \in \mathcal{J}$ and the sub-predicate y_j as well as a set of strings aux_y, which we call the auxiliary key inputs, and it outputs a subset of the original algorithm.

In particular, it should hold that:

- **Well-formed parameters:** The parameters $(n_\alpha, n_b, n_\beta, \boldsymbol{\alpha}, \mathbf{b}, \boldsymbol{\beta}) \leftarrow \text{Param}(\text{par})$ and $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{b}_i) \leftarrow \text{SplitParam}_i(\text{par}_i)$ are such that

$$(\boldsymbol{\alpha}, \mathbf{b}, \boldsymbol{\beta}) \approx (\{\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{b}_i\}_{i\in\mathcal{I}}),$$

  and for all $b_j \in \mathbf{b}$ for which there are at least two distinct $i, i' \in \mathcal{I}$ with $b_j \in \mathbf{b}_i, \mathbf{b}_{i'}$, we have $\mathcal{F}_1(b_j) > 0$.

- **Well-formed keys:** Firstly, the keys need to be well-defined, meaning that $\mathbf{k}_j(\mathbf{r}_j, \hat{\mathbf{r}}_j, \boldsymbol{\alpha}_{\mathsf{AssignMPK}(j)}, \boldsymbol{\beta}_{\mathsf{AssignMPK}(j)}, \mathbf{b}_{\mathsf{AssignMPK}(j)})$ can be generated from the subsets $\boldsymbol{\alpha}_{\mathsf{AssignMPK}(j)}$, $\boldsymbol{\beta}_{\mathsf{AssignMPK}(j)}, \mathbf{b}_{\mathsf{AssignMPK}(j)}$ and do not require other values from $\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b}$. Furthermore, the keys $(m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b})) \leftarrow \text{EncKey}(y)$ and $(\mathbf{k}_j(\mathbf{r}_j, \hat{\mathbf{r}}_j, \boldsymbol{\alpha}_{\mathsf{AssignMPK}(j)}, \boldsymbol{\beta}_{\mathsf{AssignMPK}(j)}, \mathbf{b}_{\mathsf{AssignMPK}(j)})) \leftarrow \text{SplitEncKey}_j(y_j, \text{aux}_y)$ are such that

$$\mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b}) \approx (\{\mathbf{k}_j(\mathbf{r}_j, \hat{\mathbf{r}}_j, \boldsymbol{\alpha}_{\mathsf{AssignMPK}(j)}, \boldsymbol{\beta}_{\mathsf{AssignMPK}(j)}, \mathbf{b}_{\mathsf{AssignMPK}(j)})\}_{j\in\mathcal{J}}),$$

$$\mathbf{r} \approx (\{\mathbf{r}_j\}_{j\in\mathcal{J}}), \qquad \hat{\mathbf{r}} \approx (\{\hat{\mathbf{r}}_j\}_{j\in\mathcal{J}}),$$

  and for all $r_j \in \mathbf{r}$ for which there exist at least two distinct $i, i' \in [n'_y]$ with $r_j \in \mathbf{r}_i, \mathbf{r}_{i'}$, we have $\mathcal{F}(r_j) = (\ell, \text{string})$ with $\ell > 0$ and string is derived from aux_y.

Remark 4. Any PES-ISA is a split-predicate mold for itself, where each key predicate is split in one part, as well as the parameters par.

Examples of sub-predicate spaces and aggregation function. By how we defined the key and ciphertext predicates in Section 2.5, we obtain various sub-predicate spaces for the keys and the ciphertexts (i.e., each space that is extended with a mapping is the sub-predicate space of the extended predicate space). For example, $\mathcal{Y}_{\text{CP-basic},l}$ is a sub-predicate space for each $\mathcal{A}_l \in \mathcal{AID}$ of $\mathcal{Y}_{\text{MA-CP-basic}}$. Another example is the predicate space $\mathcal{Y}_{\text{CP-single}} = \{y \in \mathcal{Y}_{\text{CP-basic}} \mid |y| = 1\}$, which is a sub-predicate space of $\mathcal{Y}_{\text{CP-basic}}$. For examples of aggregation functions, we refer to Appendices E and F.

4.3 The ISABELLA compiler

Based on the definitions from Section 3 and those above, we are now ready to give our compiler.

Compatibility. We define the notion of compatibility in the context of our compiler, which ensures that the inputs to the compiler are compatible for some PES-ISA. This covers the correctness properties for the mappings $\mathfrak{D}$ and $\mathcal{F}$. Additionally, we require that the PES-ISA uses the same prime order p as the pairing group generated by $\mathsf{PGGen}(\lambda)$ and that there exist full-domain hash functions into groups $\mathbb{G}$, $\mathbb{H}$.

Notation. In the following, we will often run the algorithms of the PES-ISA and instantiate the polynomials in the groups, i.e., we sample uniform values from $\mathbb{Z}_p$ and compute group elements whose exponents correspond to the PES-ISA polynomials evaluated on these values. To simplify notation, we will use algorithms SampleSplitParam, SampleSplitKey, SampleCt to sample those values. E.g., we write $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{b}_i)_{\mathcal{F}_1(\cdot)=0} \leftarrow \text{SampleSplitParam}_i(\text{par}_i)$, where $\mathcal{F}_1(\cdot) = 0$ indicates that we skip those variables that are instantiated with an FDH. To improve readability, we will not distinguish between formal variables and scalars. In cases where a distinction is not clear from context, we will add a note.

Definition 18 (Compiler)**.** Let $\Gamma_{\text{PES-ISA}}$ be a PES-ISA for predicate $P\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ as defined in Definition 3 with distributions $\mathfrak{D}$ and $\mathcal{F}$, and let PGGen be a pairing group generation algorithm. Further, let $\mathsf{SPM}[\Gamma_{\text{PES-ISA}}]$ be a split-predicate mold for $\Gamma_{\text{PES-ISA}}$ with $\mathcal{I}$, $\mathcal{J}$, $\mathsf{Agg}_{\text{key},\mathcal{J}}$, $\mathsf{AssignMPK}$ and sub-predicate spaces $\{\mathcal{Y}_j\}_{j\in\mathcal{J}}$ as in Definition 17. We define the scheme $\mathsf{ABE\text{-}ISA} = \mathbf{Comp}[\mathsf{PGGen}, \Gamma_{\text{PES-ISA}}, \mathsf{SPM}[\Gamma_{\text{PES-ISA}}], \mathfrak{D}, \mathcal{F}]$ as follows:

- $\mathsf{GlobalSetup}(\lambda) \to \text{GP}$: On input the security parameter λ, this algorithm generates a pairing group $\mathcal{PG} = (\mathbb{G}, \mathbb{H}, \mathbb{G}_T, p, e, g, h) \leftarrow \mathsf{PGGen}(\lambda)$, compatible with $\Gamma_{\text{PES-ISA}}$, and full-domain hash functions $\mathcal{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ and $\mathcal{H}_{\mathbb{H}} : \{0,1\}^* \to \mathbb{H}$ modeled as random oracles. It outputs global parameters $\text{GP} = (\mathcal{PG}, \mathcal{H}_{\mathbb{G}}, \mathcal{H}_{\mathbb{H}})$.

- $\mathsf{Setup}_i(\mathcal{PG}, \text{par}_i) \to (\mathsf{MPK}_i, \mathsf{MSK}_i)$: On input an index $i \in \mathcal{I}$, the group description $\mathcal{PG}$ and parameters par_i, this algorithm runs $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{b}_i)_{\mathcal{F}_1(\cdot)=0} \leftarrow \text{SampleSplitParam}_i(\text{par}_i)$, sets $\mathsf{MSK}_i = (i, \boldsymbol{\alpha}_i, \{b_j \mid b_j \in \mathbf{b}_i \wedge \mathcal{F}_1(b_j) = 0\})$ as the master secret key, and outputs MSK_i along with

$$\mathsf{MPK}_i = \big(i,\ \{[\alpha_j]_{\mathbb{G}_T}\}_{\alpha_j\in\boldsymbol{\alpha}_i},\ \{[\beta_j]_{\mathfrak{D}(\beta_j)} \mid \beta_j \in \boldsymbol{\beta_i} \wedge \mathcal{F}_1(\beta_j) = 0\},\ \{[b_j]_{\mathfrak{D}(b_j)} \mid b_j \in \mathbf{b}_i \wedge \mathcal{F}_1(b_j) = 0\}\big)$$

  as the master public key.

- $\mathsf{KeyGen}_j(\mathsf{MSK}_{\mathsf{AssignMPK}(j)}, y_j, \text{aux}_y) \to \mathsf{SK}_{y_j}$: On input a set of master secret keys $\mathsf{MSK}_{\mathsf{AssignMPK}(j)}$, a sub-predicate $y_j \in \mathcal{Y}_j$ and auxiliary input aux_y, this algorithm generates $(\mathbf{k}_j(\mathbf{r}_j, \hat{\mathbf{r}}_j, \boldsymbol{\alpha}_{\mathsf{AssignMPK}(j)}, \boldsymbol{\beta}_{\mathsf{AssignMPK}(j)}, \mathbf{b}_{\mathsf{AssignMPK}(j)})) \leftarrow \text{SplitEncKey}_j(y)$ and $(\mathbf{r}_i, \hat{\mathbf{r}}_i)_{\mathcal{F}_1(\cdot)=0} \leftarrow \text{SampleSplitKey}_j(y_j)$, and outputs the secret key SK_{y_j} as

$$\mathsf{SK}_{y_j} = \big(y_j,\ \{[r_j]_{\mathfrak{D}(r_j)} \mid j \in [m_1] \wedge \mathcal{F}_1(r_j) = 0\},\ \{[k_i]_{\mathfrak{D}(k_i)}\}_{i\in[m_3]}\big)$$

- $\mathsf{Encrypt}(\mathsf{MPK}_{\mathcal{I}_x}, x) \to (\mathsf{CT}_x, K)$: On input a set of master public keys $\mathsf{MPK}_{\mathcal{I}_x} = \{\mathsf{MPK}_i\}_{i\in\mathcal{I}_x}$, where $\mathcal{I}_x \subseteq \mathcal{I}$, some $x \in \mathcal{X}$, this algorithm first generates $(w_1, w_2, w_2', c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \mathbf{c}''(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta}) \leftarrow \text{EncCt}(x)$ and $(\mathbf{s}, \hat{\mathbf{s}}, \tilde{\mathbf{s}})_{\mathcal{F}_1(\cdot)=0} \leftarrow \text{SampleCt}(x)$, and outputs the ciphertext CT_x

$$\mathsf{CT}_x = \big(x,\ [s]_{\mathfrak{D}(s)},\ \{[s_j]_{\mathfrak{D}(s_j)} \mid j \in [w_1] \wedge \mathcal{F}_1(s_j) = 0\},$$

$$\{[c_i]_{\mathfrak{D}(c_i)}\}_{i\in[w_3]},\ \{[c_i']_{\mathbb{G}_T}\}_{i\in[w_4]},\ \{[c_i'']_{\mathfrak{D}(c_i'')}\}_{i\in[w_5]}\big)$$

and key $K = e(g,h)^{c_M}$.

- $\mathsf{Decrypt}(\mathrm{GP}, \{\mathsf{SK}_{y_j}\}_{j\in\mathcal{J}'}, \mathsf{CT}_x) \to K$: On input the global parameters GP, a set of secret keys $\{\mathsf{SK}_{y_j}\}_{j\in\mathcal{J}'}$ and a ciphertext CT_x, if $P(x,y)=1$, where $y = \mathsf{Agg}_{\mathrm{key},\mathcal{J}'}((y_j)_{j\in\mathcal{J}'})$, then this algorithm first obtains $(\mathbf{e}', \mathbf{e}'', \mathbf{E}, \overline{\mathbf{E}}) \leftarrow \mathrm{Pair}(x,y)$, sets

$$\begin{aligned}\mathfrak{P} \;=\; &\{(s_j, k_i, \mathbf{E}_{j,i}) \mid i\in[m_3], j\in\overline{[w_1]}, \mathbf{E}_{j,i}\neq 0 \wedge \mathfrak{D}(s_j)=\mathbb{G}\}\\ &\cup\{(k_i, s_j, \mathbf{E}_{j,i}) \mid i\in[m_3], j\in\overline{[w_1]}, \mathbf{E}_{j,i}\neq 0 \wedge \mathfrak{D}(s_j)=\mathbb{H}\}\\ &\cup\{(r_j, c_i, \overline{\mathbf{E}}_{i,j}) \mid i\in[w_3], j\in[m_1], \overline{\mathbf{E}}_{i,j}\neq 0 \wedge \mathfrak{D}(r_j)=\mathbb{G}\}\\ &\cup\{(c_i, r_j, \overline{\mathbf{E}}_{i,j}) \mid i\in[w_3], j\in[m_1], \overline{\mathbf{E}}_{i,j}\neq 0 \wedge \mathfrak{D}(r_j)=\mathbb{H}\}\ ,\end{aligned}$$

and

$$\mathcal{I}_{\mathbb{G}} = \{i\in[w_5] \mid \mathfrak{D}(c_i'')=\mathbb{G}\}\,, \qquad \mathcal{I}_{\mathbb{H}} = \{i\in[w_5] \mid \mathfrak{D}(c_i'')=\mathbb{H}\}$$

and then retrieves

$$\begin{aligned}&\prod_{i\in[w_4]} [c_i']_{\mathbb{G}_T}^{\mathbf{e}_i'} \prod_{i\in\mathcal{I}_{\mathbb{G}}} e([c_i'']_{\mathbb{G}}, h) \prod_{i\in\mathcal{I}_{\mathbb{H}}} e(g, [c_i'']_{\mathbb{H}}) \prod_{(\mathfrak{l},\mathfrak{r},\mathfrak{e})\in\mathfrak{P}} e([\mathfrak{l}]_{\mathbb{G}}, [\mathfrak{r}]_{\mathbb{H}})^{\mathfrak{e}}\\ &\qquad = e(g,h)^{\mathbf{e}'\mathbf{c}'^{\intercal}+\mathbf{e}''\mathbf{c}''^{\intercal}+\mathbf{s}\mathbf{E}\mathbf{k}^{\intercal}+\mathbf{c}\overline{\mathbf{E}}\mathbf{r}^{\intercal}} = e(g,h)^{c_M}.\end{aligned}$$

Remark 5. The above algorithms are defined for valid inputs. We implicitly assume that inputs are validated and that the algorithms output a failure symbol $\perp$ when given an invalid input. For example, this is the case if KeyGen_j does not get as input all MSK_i for all i contained in $\mathsf{AssignMPK}(j)$ or $y_j \notin \mathcal{Y}_j$, or if $\mathsf{Encrypt}$ does not get MPK_i necessary for $x\in\mathcal{X}$.

5 Security of our generic compiler

In this section, we define a generic security game for schemes covered by the compiler. As the compiler itself, the security game is carefully designed to argue about different levels of security. More importantly, security will be strong to enough to imply security for different types of ABE, which we will show in Section 6, and close enough to the symbolic analysis of the PES-ISA such that the heavy lifting from Section 3 can be carried over to prove security in the generic group model and from q-type assumptions.

5.1 Security model

We give a pseudocode description of the security game in Fig. 1. It captures different levels of adaptivity, depending on variables $\mathsf{X},\mathsf{Y},\mathsf{Z}\in\{\mathsf{s},\mathsf{a}\}$ which indicate whether ciphertext (challenge), secret key and corruption queries can be made adaptively or have to be announced before the actual game starts. The game draws a challenge bit δ and depending on this bit, the adversary will receive either real keys c_M or group elements chosen at random from $\mathbb{G}_T$. We will now provide more details of the exact modeling.

Master public keys and corruptions. The adversary can create master public keys adaptively via oracle MPK by specifying any $i\in\mathcal{I}$. The game runs the corresponding Setup_i algorithm, stores i in a set $\mathcal{Q}_{\mathrm{mpk}}$ and returns MPK_i. We further distinguish between static and adaptive corruptions, the former being announced via oracle InitCor. During the game, oracle Cor will then leak MSK_i. In any case, we require that the key pair was previously created using oracle MPK.

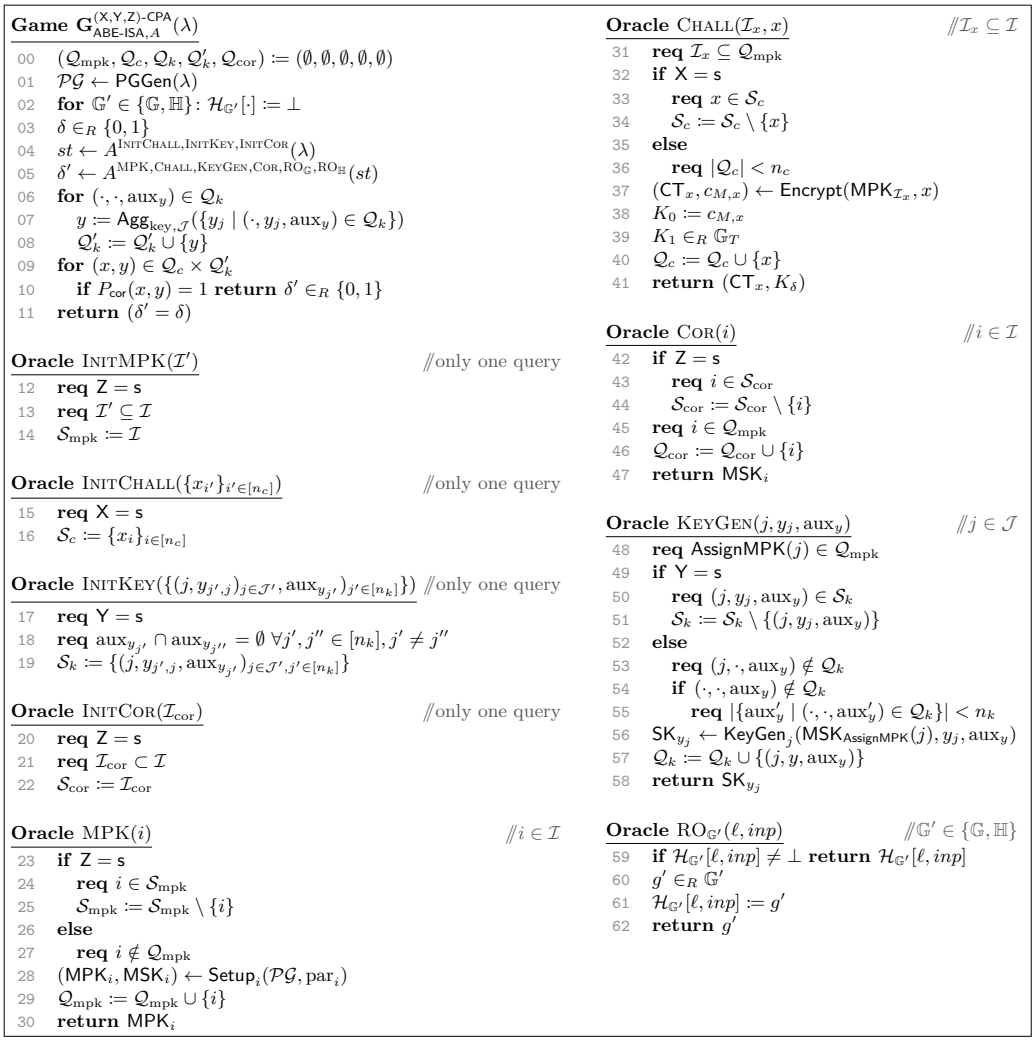


Figure 1: Security games for the class of schemes $\mathsf{ABE\text{-}ISA} = \mathbf{Comp}[\mathsf{PGGen}, \Gamma_{\text{PES-ISA}}, \mathsf{SPM}[\Gamma_{\text{PES-ISA}}], \mathfrak{D}, \mathcal{F}]$, where $\mathsf{X}, \mathsf{Y}, \mathsf{Z} \in \{\mathsf{s}, \mathsf{a}\}$, modeling static and adaptive challenge queries, secret key queries and corruptions. Integers n_c, n_k are the number of challenge and secret key queries.

One-use restriction. The security model enforces a one-use restriction on the key queries, i.e., each KeyGen_j (for $j \in \mathcal{J}$) can be queried at most once per $\mathrm{aux}_{y_{j'}}$. This is to ensure that the split keys generated from the split key generation are indistinguishable from keys directly generated with the aggregated predicate. Although it is possible that concrete constructions may remain secure even when multiple split keys are generated for the same sub-predicate space, there are examples of schemes that would break (non-trivially). Similarly, we require that all $\mathrm{aux}_{y_{j'}}$ are disjoint. In particular, if the auxiliary inputs are used to generate key components associated with non-lone variables from an FDH, using the same inputs for different key queries yields the same randomness in two keys for two different key predicates. This could be exploited to break the scheme.

Static security. The game models static security via oracles InitChall, InitKey, InitCor. These are given to the adversary (along with the security parameter λ) and each can be queried once. More specifically, the challenge oracle InitChall expects n_c

ciphertext predicates $x_{i'}$. For secret key queries, oracle InitKey expects inputs consistent with the split-predicate mold of $\Gamma_{\text{PES-ISA}}$, i.e., n_k vectors of j split-predicates $y_{j',j}$ each, where $j \in \mathcal{J}'$, along with auxiliary inputs $\text{aux}_{y_{j'}}$ for each vector. Recall that auxiliary inputs are sets of strings and here we require that they are all disjoint (cf. line 18). The announced key and ciphertext predicates are stored in sets $\mathcal{S}_c$ and $\mathcal{S}_k$. During the game, only those values will be allowed to be queried to oracles Chall and KeyGen (cf. lines 33, 50) and sets $\mathcal{Q}_c$ and $\mathcal{Q}_k$ record the queries for which ciphertexts and keys are actually provided.

Adaptive security. The game can also allow ciphertext and secret key queries to be made adaptively (when X resp. Y are set to a). In this case, only Chall and KeyGen will be available. We enforce the same one-use restriction as for static queries and additionally enforce the maximum number of queries by checking the sizes of sets $\mathcal{Q}_c$ and $\mathcal{Q}_k$ (cf. lines 36 and 55). While we also check whether the required master public keys have been created, we do not explicitly check validity of inputs (cf. Remark 5), but assume w.l.o.g. that the adversary does not issue invalid queries.

Random oracles. We also provide the adversary access to random oracles mapping into each of the source groups. Outputs are generated via lazy sampling, storing queries in lists $\mathcal{H}_{\mathbb{G}}$ and $\mathcal{H}_{\mathbb{H}}$ to provide consistent responses. The mapping $\mathcal{F}$ defines which inputs the random oracles expect and also enforce domain separation (cf. Definition 15). Again, we assume w.l.o.g. that the adversary makes only queries of this form.

Evaluating the winning condition. Eventually, the adversary terminates and outputs a bit δ'. While we have already enforced the one-use restriction, we now need to check whether for all ciphertext predicates $x_{i'}$ and key predicates $y_{j'}$, we have $P(x_{i'}, y_{j'}) = 0$. Note however, that we also need to account for trivial wins via corruptions which is why we use P_{cor} instead. The game returns whether the adversary has provided the correct challenge bit (i.e., $\delta' = \delta$) or a random bit (in case of a trivial attack).

Definition 19. Consider the games described in Fig. 1 for a scheme $\mathsf{ABE\text{-}ISA} = \mathbf{Comp}[\mathsf{PGGen}, \Gamma_{\text{PES-ISA}}, \mathsf{SPM}[\Gamma_{\text{PES-ISA}}], \mathfrak{D}, \mathcal{F}]$, adversary A and security parameter λ. We define the advantage of A in game $\mathbf{G}^{(\mathsf{X},\mathsf{Y},\mathsf{Z})\text{-}\mathsf{CPA}}_{\mathsf{ABE\text{-}ISA},A}(\lambda)$, where $\mathsf{X}, \mathsf{Y}, \mathsf{Z} \in \{\mathsf{s}, \mathsf{a}\}$, as

$$\mathsf{Adv}^{(\mathsf{X},\mathsf{Y},\mathsf{Z})\text{-}\mathsf{CPA}}_{\mathsf{ABE\text{-}ISA},A}(\lambda) \coloneqq \left| \Pr[\delta' = \delta] - \frac{1}{2} \right| .$$

Remark 6 (Adversarially generated public keys). Our game only allows corruption of honestly generated master public keys. We can easily extend the model with another oracle that allows to register adversarially generated public key material and we refer to the discussion in [AG21] for more details. Due to the strong guarantees of our SSSP, we expect that the theorem statements given below also hold in this stronger setting, when additionally relying on NIZK properties.

5.2 Static security under q-type assumptions

To prove static security, we will use a similar q-type assumption as in Ven23 [Ven23], whereas ours gives out some extra terms to support the PES extension. We call it strong (d_1, d_1', d_2)-parallel DBDH assumption and provide its definition below. Compared to the Ven23 compiler, we need these extra terms to simulate the components associated with semi-common variables $\boldsymbol{\beta}$ and non-lone ciphertext variables s_j that occur in a product with semi-common variables in $\mathbf{c}''$. Specifically, in the simulation, these terms act more like parallel DDH instances in one of the source groups than like parallel DBDH instances.

Definition 20 (The strong (d_1, d_1', d_2)-parallel DBDH assumption)**.** Let λ be the security parameter and let $\mathcal{PG} = (\mathbb{G}, \mathbb{H}, \mathbb{G}_T, p, e, g, h) \leftarrow \mathsf{PGGen}(\lambda)$ be a pairing group description. The challenger generates $\mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{c}_\mathsf{i}, \mathsf{c}'_\mathsf{j} \in_R \mathbb{Z}_p$ for all $\mathsf{i} \in [2, d_1], \mathsf{j} \in [2, d_2]$, sets $\mathsf{c}_1 = \mathsf{c}'_1 = 1$ and outputs for all $\mathbb{G}' \in \{\mathbb{G}, \mathbb{H}\}$,

$$
\begin{array}{lll}
\left[\mathsf{x}\mathsf{c}_\mathsf{i}\right]_{\mathbb{G}'}, & \text{for all } \mathsf{i} \in [d_1] \setminus \mathcal{I} & \left[\frac{\mathsf{x}\mathsf{z}\mathsf{c}_\mathsf{i}}{\mathsf{c}_{\mathsf{i}'}\mathsf{c}'_\mathsf{j}}\right]_{\mathbb{G}'}, \quad \text{for all } \mathsf{i}, \mathsf{i}' \in [d_1], \mathsf{i} \neq \mathsf{i}', \mathsf{j} \in [d_2] \\
\left[\mathsf{y}\mathsf{c}'_\mathsf{j}\right]_{\mathbb{G}'}, & \text{for all } \mathsf{j} \in [d_2] & \left[\frac{\mathsf{y}\mathsf{z}\mathsf{c}'_\mathsf{j}}{\mathsf{c}_\mathsf{i}\mathsf{c}'_{\mathsf{j}'}}\right]_{\mathbb{G}'}, \quad \text{for all } \mathsf{i} \in [d_1], \mathsf{j}, \mathsf{j}' \in [d_2], \mathsf{j} \neq \mathsf{j}' \\
\left[\frac{\mathsf{z}}{\mathsf{c}_\mathsf{i}\mathsf{c}'_\mathsf{j}}\right]_{\mathbb{G}'}, & \text{for all } \mathsf{i} \in [d_1], \mathsf{j} \in [d_2] & \\
\left[\mathsf{x}\mathsf{c}_\mathsf{i}\right]_{\mathcal{D}(\mathsf{i})}, & \text{for all } \mathsf{i} \in \mathcal{I} & \\
\left[\frac{\mathsf{y}\mathsf{z}}{\mathsf{c}_\mathsf{i}}\right]_{\mathcal{D}(\mathsf{i})}, & \text{for all } \mathsf{i} \in \mathcal{I} & \left[\frac{\mathsf{x}\mathsf{y}\mathsf{z}\mathsf{c}_\mathsf{i}}{\mathsf{c}_{\mathsf{i}'}}\right]_{\mathcal{D}(\mathsf{i})}, \quad \text{for all } \mathsf{i}, \mathsf{i}' \in \mathcal{I}, \mathsf{i} \neq \mathsf{i}'
\end{array}
$$

where $\mathcal{D}\colon \mathcal{I} \to \{\mathbb{G}, \mathbb{H}\}$ can be any distribution with $\mathcal{I} \subseteq [d_1]$ and $|\mathcal{I}| = d_1'$. By setting $\mathsf{c}_1 = \mathsf{c}'_1 = 1$, we also have that $[\mathsf{x}]_{\mathbb{G}'}, [\mathsf{y}]_{\mathbb{G}'}, [\mathsf{z}]_{\mathbb{G}'}$ are included in these terms. The challenger also flips a coin $\delta \in_R \mathbb{Z}_p$ and outputs $T \in_R \mathbb{G}_T$ if $\delta = 0$ and $T = e(g, h)^{\mathsf{x}\mathsf{y}\mathsf{z}}$ if $\delta = 1$. The adversary outputs a guess δ' for δ.

The advantage of the adversary is defined as

$$
\mathsf{Adv}^{(d_1, d_1', d_2)\text{-}\mathsf{spDBDH}}_{\mathsf{PGGen}, A}(\lambda) \coloneqq \left| \Pr[\delta' = \delta] - \frac{1}{2} \right| .
$$

The *strong* (d_1, d_1', d_2)*-parallel DBDH assumption* ((d_1, d_1', d_2)-spDBDH) holds if any PPT adversary A has at most a negligible advantage, i. e., $\mathsf{Adv}^{(d_1, d_1', d_2)\text{-}\mathsf{spDBDH}}_{\mathsf{PGGen}, A}(\lambda) \leq \mathrm{negl}(\lambda)$.

We now state our theorem capturing static security. The full proof is given in Appendix C. To provide additional confidence in the hardness of the assumption, we give concrete bounds in the generic group model in Appendix D.1.

Theorem 3. *Let $\Gamma_{PES\text{-}ISA}$ be a PES-ISA scheme that satisfies SSSP for some d_1 and d_2. Let $\mathfrak{D}$ and $\mathcal{F}$ be compatible distributions and $\mathsf{SPM}[\Gamma_{PES\text{-}ISA}]$ be a split-predicate mold for $\Gamma_{PES\text{-}ISA}$. Let $\mathsf{ABE\text{-}ISA} = \mathbf{Comp}[\mathsf{PGGen}, \Gamma_{PES\text{-}ISA}, \mathsf{SPM}[\Gamma_{PES\text{-}ISA}], \mathfrak{D}, \mathcal{F}]$ be the compiled scheme. Then, for any adversary A in the $(\mathsf{s}, \mathsf{s}, \mathsf{s})$-$\mathsf{CPA}$ game with $\mathsf{ABE\text{-}ISA}$, there exists an adversary B against (d_1, d_1', d_2)-spDBDH such that*

$$
\mathsf{Adv}^{(\mathsf{s},\mathsf{s},\mathsf{s})\text{-}\mathsf{CPA}}_{\mathsf{ABE\text{-}ISA}, A}(\lambda) \leq n_c \cdot \mathsf{Adv}^{(d_1, d_1', d_2)\text{-}\mathsf{spDBDH}}_{\mathsf{PGGen}, B}(\lambda) ,
$$

where d_1' is implied by the number of non-lone CT variables that are paired with semi-common variables. If $\mathcal{F}$ specifies that variables are generated by full-domain hashes (i.e., $\mathcal{F}_1$ maps them to nonzero), these hash functions are modeled as random oracles.

Further, if $\mathsf{SPM}[\Gamma_{PES\text{-}ISA}]$ does not use aux_y to generate (non-)lone key variables, then we also get $(\mathsf{s}, \mathsf{a}, \mathsf{s})$-$\mathsf{CPA}$ security.

Proof sketch. To prove static security from the (d_1, d_1', d_2)-spDBDH assumption, we construct a reduction that uses an adversary against static security to break the (d_1, d_1', d_2)-spDBDH assumption. Our reduction follows a hybrid argument over the challenges, replacing each encapsulated key by a random key in a series of games. For each game hop, we embed an instance of the (d_1, d_1', d_2)-spDBDH assumption in the (public and queried secret) keys and the challenge ciphertext, using the substitution matrices and vectors in the SSSP proof to cancel out the terms that we cannot create.

Roughly, we encode the terms $\left[\mathsf{y}\mathsf{c}'_\mathsf{j}\right]_{\mathbb{G}'}$ in the secret keys associated with the non-lone key variables, the terms $\left[\frac{\mathsf{z}}{\mathsf{c}_\mathsf{i}\mathsf{c}'_\mathsf{j}}\right]_{\mathbb{G}'}$ in the public keys associated with the common variables and we pair $\left[\frac{\mathsf{z}}{\mathsf{c}_\mathsf{i}}\right]_{\mathbb{G}'}$ and $\left[\mathsf{y}\mathsf{c}'_\mathsf{j}\right]_{\mathbb{G}'}$ to obtain the proper substitutions for the public keys

associated with the master-key variables. For the non-lone ciphertext variables, we use $[\mathsf{xc_i}]_{\mathbb{G}'}$ if they are not paired with semi-common variables. If they are paired, then we use the restricted terms $[\mathsf{xc_i}]_{\mathcal{D}(\mathsf{i})}$. Similarly, we use the restricted terms $\left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]_{\mathcal{D}(\mathsf{i})}$ for the (associated) semi-common variables. Note that the indices i and j are used to encode the substitution vectors and matrices, e.g., embedding the vector entry $(\mathbf{r}_j)_\mathsf{j}$ requires us to use the term $\left[\mathsf{yc}'_\mathsf{j}\right]_{\mathbb{G}'}$.

If we encode the non-lone variables and public-key variables in this way, the key and ciphertext components associated with the polynomials can be programmed using the terms $\left[\frac{\mathsf{xzc_i}}{\mathsf{c_{i'}c'_j}}\right]_{\mathbb{G}'}$ (for the ciphertext polynomials $\mathbf{c}$), the terms $\left[\frac{\mathsf{yzc}'_\mathsf{j}}{\mathsf{c_i c}'_{\mathsf{j}'}}\right]_{\mathbb{G}'}$ (for the key polynomials), the terms $\left[\frac{\mathsf{xyzc_i}}{\mathsf{c_{i'}}}\right]_{\mathcal{D}(\mathsf{i})}$ (for the ciphertext polynomials $\mathbf{c}''$) and, by pairing the terms $[\mathsf{xc_i}]_{\mathbb{G}'}$ and $\left[\frac{\mathsf{yzc}'_\mathsf{j}}{\mathsf{c_i c}'_{\mathsf{j}'}}\right]_{\mathbb{G}'}$, we can program the ciphertexts associated with the polynomials $\mathbf{c}'$. Note that these terms can be used to program the components associated with the products among the vectors and matrices that do not follow an actual matrix-vector product. For example, the product of $\mathbf{r}_j$ and $\mathbf{B}_k$ in the encodings yields $\prod_{\mathsf{i},\mathsf{j},\mathsf{j}'}(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{r}_j)_{\mathsf{j}'}$, whereas a real product of the matrix $\mathbf{B}_k$ and $\mathbf{r}_j$ yields the product $\prod_{\mathsf{i},\mathsf{j}}(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{r}_j)_\mathsf{j}$. Hence, we use the aforementioned terms for $\mathsf{j} \neq \mathsf{j}'$ (which do not follow the matrix-vector product) and we use the SSSP proof to cancel out the terms for $\mathsf{j} = \mathsf{j}'$ (which do follow the matrix-vector product).

Lastly, we need to give the adversary access to the associated secret keys of the corrupted public-key components. We use here the fact that the substitution vectors and matrices for the master-key, semi-common and common variables are all-zero because of the SSSP proof. This ensures that we do not encode any terms of the (d_1, d'_1, d_2)-spDBDH assumption in those components. Hence, we know the exponents of those components (i.e., $\bar{\alpha}_j$, $\bar{\beta}_j$ and $\bar{b}_k$) and can give those to the adversary. □

5.3 Adaptive security in the GGM

Similar to previous works [ABGW17, AG21, RW22], we prove security in the generic group model (GGM). For an overview of the GGM we refer to Appendix D. We improve upon previous works by additionally considering adaptive corruptions, using our carefully designed notion of strong MK-MC security under corruptions (cf. Definition 12). This gives us the following statement.

Theorem 4. *Let $\Gamma_{PES\text{-}ISA}$ be a PES-ISA scheme that satisfies SSSP. Let $\mathfrak{D}$ and $\mathcal{F}$ be compatible distributions and $\mathsf{SPM}[\Gamma_{PES\text{-}ISA}]$ be a split-predicate mold for $\Gamma_{PES\text{-}ISA}$. Let $\mathsf{ABE\text{-}ISA} = \mathbf{Comp}[\mathsf{PGGen}, \Gamma_{PES\text{-}ISA}, \mathsf{SPM}[\Gamma_{PES\text{-}ISA}], \mathfrak{D}, \mathcal{F}]$ be the compiled scheme. Let A be a generic adversary in game $(\mathsf{a},\mathsf{a},\mathsf{a})$-CPA game with ABE-ISA, issuing at most n_{mpk} queries to MPK, n_c queries to Chall, n_k queries to KeyGen with distinct aux_y, n_{ro} queries to both random oracles, n_{op} queries to Op and n_{pair} queries to Pair. Then,*

$$\mathsf{Adv}^{(\mathsf{a},\mathsf{a},\mathsf{a})\text{-}\mathsf{CPA}}_{\mathsf{ABE\text{-}ISA},A}(\lambda) \leq \frac{4n^2}{p} \ ,$$

where $n \leq 2 + n_{\text{mpk}}|\mathsf{MPK}| + n_c|\mathsf{CT}| + n_k|\mathsf{SK}| + n_{\text{ro}} + n_{\text{op}} + n_{\text{pair}}$ with $|\mathsf{MPK}| = \max\{|\mathsf{MPK}_i| \mid i \in \mathcal{Q}_{mpk}\}$, $|\mathsf{CT}| = \max\{|\mathsf{CT}_x| \mid x \in \mathcal{Q}_c\}$ and $|\mathsf{SK}| = \max\{|\mathsf{SK}_y| \mid y \in \mathcal{Q}'_k\}$.

If $\mathcal{F}$ specifies that variables are generated by full-domain hashes (i.e., $\mathcal{F}_1$ maps them to nonzero), these hash functions are modeled as random oracles.

We provide the full proof in Appendix D.2 and sketch the main ideas below. Assuming that n is dominated by the number of group operation queries n_{op} and random oracle queries n_{ro} (because these can be considered offline computation), the bound matches that

of breaking the discrete logarithm problem up to a small constant factor and is therefore optimal.

Proof sketch. The adversary in the GGM has access to the oracles defined in the security game, as well as a group operation and pairing oracle. In order to perform group operations, the challenger provides the adversary with unique (and representation-independent) handles to them. We want to show that the simulation in the GGM is indistinguishable from a symbolic execution, where the challenger instead uses polynomials to represent and determine equality of group elements. Luckily, the PES scheme $\Gamma_{\text{PES-ISA}}$ already uses polynomials to describe exactly what is happening in the exponents of master public keys, ciphertexts and secret keys. Using the group operation and pairing oracle, the adversary can further combine group elements. However, by strong MK-MC security of the PES (Definition 7), these combinations do not help the adversary in breaking security. Therefore, in the symbolic model the challenge is independent of the secret bit δ and we only need to bound the difference between the symbolic simulation and the simulation in the GGM. For this we look at the probability that two polynomials are different, but their evaluation is the same. That is, when replacing formal variables with uniformly random elements from $\mathbb{Z}_p$, we might produce a collision. We can bound the probability using the Schartz-Zippel Lemma (Lemma 13) and the maximum number of group elements n in the simulation.

So far, we have ignored that the adversary can also corrupt master public keys. It turns out that the above approach fails when considering adaptive corruptions because they reveal exponents to the adversary which can be used in future queries. Therefore, instead of applying the Schwartz-Zippel Lemma once, we have to use it more carefully after each corruption. In particular, the challenger assigns a random value to the formal variable that the adversary asks to corrupt and then it checks whether its simulation is still perfect or whether a collision occurred. In the latter case, we abort. When the adversary terminates and outputs its guess, we rely on strong MK-MC security under corruptions (Definition 12) to argue that the adversary's view is independent of the challenge bit. Then we carefully sum up the probabilities to get the bound in the theorem. □

6 Applications and new schemes

In this section, we demonstrate how our compiler can be used to construct schemes with several types of advanced functionalities from pair encodings. It is straightforward to see that, if we take the trivial mold for a PES-ISA, we get a regular ABE scheme.

6.1 Multi-authority attribute-based encryption

We can create multi-authority schemes in a more modular way using our compiler. In particular, we would take as input a PES for multi-authority predicate spaces $\mathcal{X}_{\text{MA}}$ and $\mathcal{Y}_{\text{MA}}$ with auxiliary key inputs aux_y containing a global identifier GID, such that we also have a split-predicate mold for authority-independent sub-predicate spaces for $\mathcal{Y}_{\text{MA}}$. Using this PES and mold as input, our compiler yields a scheme that can be directly used to build an ABE scheme for the multi-authority ABE syntax. As we show below, the difference between the scheme generated with our compiler and the scheme following the MA-ABE syntax is purely syntactical, and thus, the resulting scheme is immediately secure if the scheme created with our compiler is secure.

Generic construction. Let PES-ISA = (Param, EncKey, EncCt, Pair) be a PES for a multi-authority predicate $P\colon \mathcal{X}_{\text{MA}} \times \mathcal{Y}_{\text{MA}} \to \{0,1\}$ with compatible encodings specified by $\mathfrak{D}$ and $\mathcal{F}$, such that SSSP holds for some d_1 and d_2. Suppose that PES-ISA has a

split-predicate mold with sub-predicate spaces $\mathcal{Y}_{\text{SA}}$ and auxiliary key inputs aux_y that includes GID, such that each $y \in \mathcal{Y}_{\text{MA}}$ associated with aux_y can be split in $\{y_{\mathsf{GID},\mathcal{A}_j} \mid j \in \mathcal{J}'\}$ with $\mathcal{J}' \subseteq [n_{\text{aut}}]$. Let $\mathsf{ABE} = \mathbf{Comp}[\mathcal{PG}, \text{PES-ISA}, \mathsf{SPM}[\Gamma_{\text{PES-ISA}}], \mathfrak{D}, \mathcal{F}]$ be the ABE scheme generated via the ISABELLA compiler, with algorithms $\mathsf{ABE} = (\mathsf{GlobalSetup}_{\mathsf{ABE}}, \{\mathsf{Setup}_{\mathsf{ABE},\mathcal{A}_i}\}_{i\in[n_{\text{aut}}]}, \{\mathsf{KeyGen}_{\mathsf{ABE},\mathcal{A}_j} \mid j \in [n_{\text{aut}}]\}, \mathsf{Encrypt}_{\mathsf{ABE}}, \mathsf{Decrypt}_{\mathsf{ABE}})$. Then, we construct a multi-authority scheme MA-ABE from the compiled scheme ABE using the proper syntax as follows:

- $\mathsf{GlobalSetup}(\lambda)$: The algorithm returns the global parameters $\text{GP} = (\mathcal{PG}, \mathcal{H}_{\mathbb{G}}, \mathcal{H}_{\mathbb{H}}) \leftarrow \mathsf{GlobalSetup}_{\mathsf{ABE}}(\lambda)$.
- $\mathsf{AuthoritySetup}(\text{GP})$: The algorithm computes authority key pair $(\mathsf{MPK}_{\mathcal{A}}, \mathsf{MSK}_{\mathcal{A}}) \leftarrow \mathsf{Setup}_{\mathsf{ABE},\mathcal{A}}(\mathcal{PG}, \text{par}_{\mathcal{A}})$ and returns $(\mathcal{A}, \mathsf{MPK}_{\mathcal{A}}, \mathsf{MSK}_{\mathcal{A}})$.
- $\mathsf{KeyGen}(\mathcal{A}, \mathsf{MSK}_{\mathcal{A}}, \mathsf{GID}, y_{\mathsf{GID},\mathcal{A}})$: The algorithm returns the key $\mathsf{SK}_{\mathsf{GID},\mathcal{A},y_{\mathsf{GID},\mathcal{A}}} \leftarrow \mathsf{KeyGen}_{\mathsf{GID},\mathcal{A}}(\mathsf{MSK}_{\mathsf{GID},\mathcal{A}}, y_{\mathsf{GID},\mathcal{A}}, \text{aux}_y)$, where aux_y contains GID.
- $\mathsf{Encrypt}(\{\mathcal{A}_i, \mathsf{MPK}_{\mathcal{A}_i}\}, x)$: The algorithm returns $(\mathsf{CT}_x, K) \leftarrow \mathsf{Encrypt}_{\mathsf{ABE}}(\{\mathsf{MPK}_{\mathcal{A}_i}\}_i, x)$.
- $\mathsf{Decrypt}(\text{GP}, \{\mathsf{SK}_{\mathsf{GID},\mathcal{A}_j,y_{\mathsf{GID},\mathcal{A}_j}}\}_{j\in\mathcal{J}'}, \mathsf{CT}_x)$: The algorithm returns the decapsulated key K output by $\mathsf{Decrypt}(\text{GP}, \{\mathsf{SK}_{\mathsf{GID},\mathcal{A},y_{\mathsf{GID},\mathcal{A}}}\}_{j\in\mathcal{J}'}, \mathsf{CT}_x)$.

Corruptions and P_{cor}. For MA-ABE, we can describe P_{cor} as follows. In particular, corrupting an authority automatically sets a sub-predicate for that authority to true. For example, for CP-ABE, the predicate $P_{\text{MA-CP-basic}} \colon \mathcal{X}_{\text{MA-CP-basic}} \times \mathcal{Y}_{\text{MA-CP-basic}} \to \{0,1\}$ is defined as $P_{\text{MA-CP-basic}}((\mathbf{A}, \rho, \tilde{\rho}), \mathcal{S}) = 1$ if and only if there exists $\Upsilon = \{j \in [n_1] \mid (\tilde{\rho}(j), \rho(j)) \in \mathcal{S}\}$ and $\{\varepsilon_j\}_{j\in\Upsilon}$ such that $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$. Let $\mathfrak{C} \subseteq \tilde{\rho}([n_1])$ denote a set of corrupted authorities. Then the corruptable-variable predicate P_{cor} is defined as $P_{\mathsf{cor}}((\mathbf{A}, \rho, \tilde{\rho}), \mathcal{S}) = 1$ if and only if there exists $\Upsilon = \{j \in [n_1] \mid \tilde{\rho}(j) \in \mathfrak{C} \vee (\tilde{\rho}(j), \rho(j)) \in \mathcal{S}\}$ and $\{\varepsilon_j\}_{j\in\Upsilon}$ such that $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, ..., 0)$.

Corollary 2. *The multi-authority scheme* MA-ABE *is statically secure assuming* (d_1, d_1', d_2)-spDBDH*, where* d_1 *and* d_2 *are as in the proof for SSSP. Further,* MA-ABE *is fully secure in the GGM.*

Proof. This follows directly from Theorems 3 and 4 and by observing that the changes to ABE are purely syntactic. □

One-use restriction. There is a one-use restriction on the key generation oracle in that KeyGen can be queried only once per $(\mathcal{A}, \mathsf{GID})$ combination that must technically also be enforced in practice. For some schemes, it seems that the scheme would not break if the key generation is queried multiple times on the same inputs because either the key is generated completely deterministically or provides sufficient randomness to be secure even if multiple instances of attributes occur, but it is strictly speaking not proven that it is secure to do so.

Existing schemes and new schemes. In Appendices E and F, multiple constructions of PES-ISA are given, for which we describe explicit molds, that allow for multi-authority support. In particular, Definition 23 is a new multi-authority KP-ABE scheme that allows the encrypting user to pick the set of authorities for which the decrypting user needs to have a key. The scheme in Definition 24 is a new decentralized CP-ABE scheme that supports negations in a completely unbounded fashion. This is the first of its kind.

Demonstrating our framework via an example. Furthermore, to demonstrate our framework, we show how to build a full-fledged (multi-authority) ABE scheme from pair encodings using our methodology. Specifically, we give descriptions of $\mathcal{F}$ and $\mathfrak{D}$, and describe the SPM for multi-authority ABE. In addition, to illustrate that including $\boldsymbol{\beta}$ and $\mathbf{c}''$ in our scheme improves the efficiency of multi-authority schemes, we have included an efficiency analysis of this scheme, and compare its instantiation with the ISABELLA compiler with its instantation with the Ven23 compiler. We also compare it with other state-of-the-art decentralized schemes to show that the resulting scheme compares favorably with those as well.

6.2 Functional adaptivity in ABE

We introduce the new notion of functional adaptivity in the context of ABE, which is intrinsic in our compiler definition. At a high level, functional adaptivity allows, e.g., secret keys, to be generated adaptively rather than in one key query. Instead of generating a key for the entire key predicate, functional adaptivity allows us to split this process in phases. For example, for regular CP-ABE (with predicate spaces $\mathcal{X}_{\text{CP-basic}}$ and $\mathcal{Y}_{\text{CP-basic}}$), we would be able to generate keys in an attribute-wise fashion [VAH23] rather than for an entire set. In this way, users can request secret keys for attributes that are used in ciphertexts but for which they have not yet obtained a key without having to obtain keys for the entire updated set. This mitigates the computational effort required from the authority, and could also improve the resilience of the system. Oftentimes, some form of key rotation is deployed to ensure that keys can be revoked once attributes have changed [LVV+23]. Rather than renewing the entire key, it would be possible to renew only parts of the key, affecting fewer users in the same system, therefore reducing the impact both on the user and authority side. We provide more details in the paragraphs below.

CP-ABE with attribute-wise key generation. We define a general mold for CP-ABE with attribute-wise key generation. Let $\mathcal{Y}_{\text{CP-var}}$ be a key predicate space for the attribute universe $\mathcal{U}$ such that each $\mathbf{y} \in \mathcal{Y}_{\text{CP-var}}$ is a tuple (of length $\ell \geq 1$) with, without loss of generality, set $\mathcal{S} \in \mathcal{Y}_{\text{CP-basic}}$ in one of the entries. We define, for each $\mathsf{att} \in \mathcal{U}$, the sub-predicate space $\mathcal{Y}_{\text{CP-var-sub},\mathsf{att}} = \{(\mathsf{att}, (\mathbf{y})_{[2,\ell]}) \mid \mathbf{y} \in \mathcal{Y}_{\text{CP-var}}\}$. Hence, each $\mathbf{y} \in \mathcal{Y}_{\text{CP-var}}$ (where $(\mathbf{y})_1 = \mathcal{S}$) can be split in sub-predicates $(\mathsf{att}, (\mathbf{y})_{[2,\ell]})$ for all $\mathsf{att} \in \mathcal{S}$. Let $\Gamma_{\text{PES-ISA}}$ be a PES-ISA for some predicate $P_{\text{CP-var}} \colon \mathcal{X}_{\text{CP-var}} \times \mathcal{Y}_{\text{CP-var}} \to \{0,1\}$. If there exists a split-predicate mold for $\Gamma_{\text{PES-ISA}}$ with these sub-predicate spaces and auxiliary key inputs aux_y that include a global identifier GID, then our compiler yields a CP-ABE scheme for the original predicate with attribute-wise key generation. To distinguish the key generation of a scheme with a "regular" key generation from a scheme with a "attribute-wise" key generation, we change the syntax of the key generation algorithm from KeyGen to $\mathsf{AttrKeyGen}$, and the algorithm takes as input the sub-predicates $(\mathsf{att}, (\mathbf{y})_{[2,\ell]})$ for the keys rather than the whole key predicate. In Appendix E.2, we give an example of a CP-ABE scheme with attribute-wise key generation.

Functional adaptivity in schemes that support negations. For some types of predicates, we can support only a limited type of functional adaptivity (as also mentioned in [VAH23]). For example, for large-universe CP-ABE schemes that support negations in the policies, we cannot support an attribute-wise key generation, because the mechanism that is in place to support negations is incompatible with an attribute-wise key generation support. Roughly, there exist three mechanisms to support negations:

- **Using negative attributes:** which implements a negation by considering negative attributes;

- **Comparing the whole set:** which compares the whole set with the negated attribute to verify that the negated attribute is not in the set;

- **Labeled comparison with the set:** which compares only the attributes in the set with the same label as the negated attribute, and there must be at least one attribute in the set that has the same label as the negated attribute.

For the first type, it is strictly speaking not possible to support it in the large-universe setting, because each user would require a secret key for an exponential number of attributes. Nevertheless, an attribute-wise key generation as defined in our paper could support a type of "lazy sampling", where a user would request a key for a negative attribute once this is needed. However, we argue that it difficult to enforce securely and efficiently in practice. Most notably, to implement this securely, the authority would need to keep a database of all the attributes (positive and negative instances) for which it has issued keys to a user. If the authority does not do this, then it could issue a key for both the positive and the negative instance of the attribute, breaking the security with respect to this attribute. This database's size would scale in the number of users and in the number of positive and negative attributes, which may be impractical.

For the other two types, it is impossible to support it. In the first place, concrete schemes that support this type of negations do not have the structural independence that we require for the split-predicate algorithms. More generally, it also seems impossible to support attribute-wise key generation in schemes that support this type of negations, because the required independence of the "attribute-related keys" implies a trivial attack on this type of negation. That is, we could simply "leave out" the negated attribute from our set of "attribute-related keys" when decrypting. To protect against such attacks, schemes that support this type of negations strongly link the keys together so that it is impossible to simply "leave out" the negated attribute.

Although none of the non-monotone schemes can support an attribute-wise key generation (at all or efficiently), we argue that schemes of the third type can support a label-wise key generation, which allows for a better balance between the two addressed issues. Roughly, the idea is that users can request keys for those attributes that they possess whose labels are the same. Hence, users issue a set of keys for one label in each key request instead of the whole set. We show that schemes that support this type of negation satisfy the structural requirements posed by the split-predicate mold. Furthermore, to enforce this type of negation securely in practice, the authority does not need to store all the attributes for which it issues keys, only the labels. Not only does this scale better, but this could also provide better privacy properties, for example, if the attributes may contain sensitive information.

CP-ABE with label-wise key generation. We define the notion of CP-ABE with label-wise key generation for CP-ABE that defines an explicit mapping for the labels associated with the attributes, i.e., which have predicate spaces $\mathcal{X}_{\text{CP-var-}\rho_{\mathsf{lab}}}$ and $\mathcal{Y}_{\text{CP-var-}\rho_{S,\mathsf{lab}}}$ (where the labeling for the key space may be implicit, like in [Ven23] and in Definition 24)). We define, for each label lab, the sub-predicate space $\mathcal{Y}_{\text{CP-var},\mathsf{lab}} = \{(\mathsf{lab}, y_{\mathsf{lab}}) \mid y_{\mathsf{lab}} \in \mathcal{Y}_{\text{CP-var}}\}$. If there exists a split-predicate mold for $\Gamma_{\text{PES-ISA}}$ with these sub-predicate spaces and auxiliary key inputs aux_y that include a global identifier GID, then our compiler yields a CP-ABE scheme for the original predicate with label-wise key generation. We show that the new decentralized CP-ABE scheme in Appendix F.2 can support a label-wise key generation by giving a split-predicate mold for this functionality.

Acknowledgments

The authors would like to honor the memory of Antonio de la Piedra, who worked on the ACABELLA extension in its early stages. The authors would also like to thank Sven Argo for some discussions on the Ven23 compiler that have influenced the design of our compiler. Doreen Riepel was supported in part by Bellare's KACST grant.

References

[ABGW17] M. Ambrona, G. Barthe, R. Gay, and H. Wee. Attribute-based encryption in the generic group model: Automated proofs and new constructions. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *CCS*, pages 647–664. ACM, 2017.

[AC16] S. Agrawal and M. Chase. A study of pair encodings: Predicate encryption in prime order groups. In E. Kushilevitz and T. Malkin, editors, *TCC*, volume 9563 of *LNCS*, pages 259–288. Springer, 2016.

[AC17a] S. Agrawal and M. Chase. FAME: fast attribute-based message encryption. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *CCS*, pages 665–682. ACM, 2017.

[AC17b] S. Agrawal and M. Chase. Simplifying design and analysis of complex predicate encryption schemes. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT*, volume 10210 of *LNCS*, pages 627–656. Springer, 2017.

[AG21] M. Ambrona and R. Gay. Multi-authority abe, revisited. Cryptology ePrint Archive, Paper 2021/1381, 2021.

[AG23] M. Ambrona and R. Gay. Multi-authority ABE for non-monotonic access structures. In A. Boldyreva and V. Kolesnikov, editors, *PKC*, volume 13941 of *LNCS*, pages 306–335. Springer, 2023.

[AGM+] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[AGM+13] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptogr. Eng.*, 3(2):111–128, 2013.

[Amb21] M. Ambrona. Generic negation of pair encodings. In J. A. Garay, editor, *PKC*, volume 12711 of *LNCS*, pages 120–146. Springer, 2021.

[AT20] N. Attrapadung and J. Tomida. Unbounded dynamic predicate compositions in ABE from standard assumptions. In *ASIACRYPT*, pages 405–436. Springer, 2020.

[Att14] N. Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *LNCS*, pages 557–577. Springer, 2014.

[Att19] N. Attrapadung. Unbounded dynamic predicate compositions in attribute-based encryption. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT*, volume 11476 of *LNCS*, pages 34–67. Springer, 2019.

[Bei96] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. Phd thesis, Ben Gurion University, 1996.

[BFP21] Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 587–617. Springer, Heidelberg, December 2021.

[BGMW92] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation (extended abstract). In R. A. Rueppel, editor, *EUROCRYPT*, volume 658 of *LNCS*, pages 200–207. Springer, 1992.

[BLS02] P S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN*, volume 2576 of *LNCS*, pages 257–267. Springer, 2002.

[Bow] S. Bowe. BLS12-381: New zk-SNARK elliptic curve construction. https://blog.z.cash/new-snark-curve/.

[CC09] M. Chase and S. S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *CCS*, pages 121–130. ACM, 2009.

[CGW15] J. Chen, R. Gay, and H. Wee. Improved dual system ABE in prime-order groups via predicate encodings. In E. Oswald and M. Fischlin, editors, *EUROCRYPT*, volume 9057 of *LNCS*, pages 595–624. Springer, 2015.

[Cha07] M. Chase. Multi-authority attribute-based encryption. In S. P. Vadhan, editor, *TCC*, volume 4392 of *LNCS*, pages 515–534. Springer, 2007.

[DKW23a] P. Datta, I. Komargodski, and B. Waters. Decentralized multi-authority abe for nc^1 from computational-bdh. *J. Cryptol.*, 36(2):6, 2023.

[DKW23b] P. Datta, I. Komargodski, and B. Waters. Fully adaptive decentralized multi-authority ABE. In C. Hazay and M. Stam, editors, *EUROCRYPT*, volume 14006 of *LNCS*, pages 447–478. Springer, 2023.

[dlPVA22] A. de la Piedra, M. Venema, and G. Alpár. ABE squared: Accurately benchmarking efficiency of attribute-based encryption. *TCHES*, 2022(2):192–239, 2022.

[dlPVA23] A. de la Piedra, M. Venema, and G. Alpár. ACABELLA: automated (crypt)analysis of attribute-based encryption leveraging linear algebra. In W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, editors, *CCS*, pages 3269–3283. ACM, 2023.

[EHK+13] A. Escala, G. Herold, E. Kiltz, C. Ràfols, and J. L. Villar. An algebraic framework for diffie-hellman assumptions. In R. Canetti and J. A. Garay, editors, *CRYPTO*, volume 8043 of *LNCS*, pages 129–147. Springer, 2013.

[ETS18] ETSI. ETSI TS 103 458 (V1.1.1). Technical specification, European Telecommunications Standards Institute (ETSI), 2018.

[GPSW06a] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *CCS*. ACM, 2006.

[GPSW06b] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. Cryptology ePrint Archive, Report 2006/309, 2006.

[Gui20] A. Guillevic. Pairing-friendly curves. https://members.loria.fr/AGuillevic/pairing-friendly-curves/, 2020.

[KL10] S. Kamara and K. E. Lauter. Cryptographic cloud storage. In R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. M. Miret, K. Sako, and F. Sebé, editors, *RLCPS*, volume 6054 of *LNCS*, pages 136–149. Springer, 2010.

[KPRR23] Eike Kiltz, Jiaxin Pan, Doreen Riepel, and Magnus Ringerud. Multi-user CDH problems and the concrete security of NAXOS and HMQV. In Mike Rosulek, editor, *CT-RSA 2023*, volume 13871 of *LNCS*, pages 645–671. Springer, Heidelberg, April 2023.

[LHC+11] J. Li, Q. Huang, X. Chen, S. S. M. Chow, D. S. Wong, and D. Xie. Multi-authority ciphertext-policy attribute-based encryption with accountability. In B. S. N. Cheung, L. Chi Kwong Hui, R. S. Sandhu, and D. S. Wong, editors, *ASIACCS*, pages 386–390. ACM, 2011.

[LVV+23] W. Ladd, T. Verma, M. Venema, A. Faz-Hérnandez, B. McMillion, A. Wildani, and N. Sullivan. Portunus: Re-imagining access control in distributed systems. In J. Lawall and D. Williams, editors, *USENIX ATC*, pages 35–52. USENIX Association, 2023.

[LW11] A. Lewko and B. Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588. Springer, 2011.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005.

[MJ18] Y. Michalevsky and M. Joye. Decentralized policy-hiding ABE with receiver privacy. In J. López, J. Zhou, and M. Soriano, editors, *ESORICS*, volume 11099 of *LNCS*, pages 548–567. Springer, 2018.

[Nat23] National Institute of Standards and Technology. NIST IR 8214C – NIST First Call for Multi-Party Threshold Schemes. https://csrc.nist.gov/pubs/ir/8214/c/ipd, 2023.

[RVV24] Doreen Riepel, Marloes Venema, and Tanya Verma. ISABELLA: Improving Structures of Attribute-Based Encryption Leveraging Linear Algebra. In *CCS*. ACM, 2024.

[RW13] Y. Rouselakis and B. Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *CCS*, pages 463–474. ACM, 2013.

[RW15] Y. Rouselakis and B. Waters. Efficient statically-secure large-universe multi-authority attribute-based encryption. In R. Böhme and T. Okamoto, editors, *FC*, volume 8975 of *LNCS*, pages 315–332. Springer, 2015.

[RW22] D. Riepel and H. Wee. FABEO: fast attribute-based encryption with optimal security. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *CCS*, pages 2491–2504. ACM, 2022.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *EUROCRYPT*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.

[SRGS12] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In *USENIX Security Symposium*, pages 175–188. USENIX Association, 2012.

[SW05] A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 457–473. Springer, 2005.

[TKN20] J. Tomida, Y. Kawahara, and R. Nishimaki. Fast, compact, and expressive attribute-based encryption. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC*, volume 12110 of *LNCS*, pages 3–33. Springer, 2020.

[VA] M. Venema and G. Alpár. Performance estimates for the GLUE paper. https://github.com/mtcvenema/glue.

[VA22] M. Venema and G. Alpár. TinyABE: Unrestricted ciphertext-policy attribute-based encryption for embedded devices and low-quality networks. In L. Batina and J. Daemen, editors, *AFRICACRYPT*, volume 13503 of *LNCS*, pages 103–129. Springer, 2022.

[VA23] M. Venema and G. Alpár. GLUE: generalizing unbounded attribute-based encryption for flexible efficiency trade-offs. In A. Boldyreva and V. Kolesnikov, editors, *PKC*, volume 13940 of *LNCS*, pages 652–682. Springer, 2023.

[VAH23] M. Venema, G. Alpár, and J.-H. Hoepman. Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. *Des. Codes Cryptogr.*, 91(1):165–220, 2023.

[VB24] M. Venema and L. Botros. Using predicate extension for predicate encryption to generically obtain chosen-ciphertext security and signatures. *IACR Communications in Cryptology*, 1(1), 2024.

[Ven23] M. Venema. A practical compiler for attribute-based encryption: New decentralized constructions and more. In M. Rosulek, editor, *CT-RSA*, volume 13871 of *LNCS*, pages 132–159. Springer, 2023.

[Wee14] H. Wee. Dual system encryption via predicate encodings. In Y. Lindell, editor, *TCC*, volume 8349 of *LNCS*, pages 616–637. Springer, 2014.

[WMZV16] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan. Sieve: Cryptographically enforced access control for user data in untrusted clouds. In Katerina J. Argyraki and Rebecca Isaacs, editors, *NSDI*, pages 611–626. USENIX Association, 2016.

[ZZ23] Cong Zhang and Mark Zhandry. The relationship between idealized models under computationally bounded adversaries. LNCS, pages 390–419. Springer, Heidelberg, December 2023. doi:10.1007/978-981-99-8736-8_13.

A The ACABELLA extension

For the automated proofs, we expand the ACABELLA framework [dlPVA23], which currently covers only PES-AC17. In this section, we describe all the algorithms on which our extension of the ACABELLA tool is based.

A.1 Our parser and correctness checker

On the representation of the encodings. For the proof generation, it is beneficial to consider a simpler representation of the encodings than Definition 3, and let the tool derive assignment for the variables (e.g., master-key/semi-common/common/(non-)lone key/(non-)lone ciphertext) and encodings (variable or polynomial). Furthermore, instead of distinguishing between master-key variables $\boldsymbol{\alpha}$ and semi-common variables $\boldsymbol{\beta}$, and non-primed and primed ciphertext encodings over $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$, we can treat them the same in the proof generation. (This is because the proof generation for these variables follows the same strategy. The only things that distinguishes these values is in which groups they are instantiated, which is in part influenced by whether they are generated by a full-domain hash or not.)

The inputs. The tool gets as input:

- The master public key encodings **abenc**;
- The key encodings **kenc**;
- The ciphertext encodings **cenc**;
- The blinding/target value: c_M;
- A set of known variables.

The goal is to parse the inputs in multiple passes over the inputs.

The first pass: distinguishing variables from polynomials. In the first pass over the inputs, we first make a split in the key and ciphertext encodings:

- The key encodings **kenc** are split in $\mathbf{r}$ and $\mathbf{k}$, where $\mathbf{r}$ consists of non-lone variables that occur only in the keys (and not in the master public key encodings) and $\mathbf{k}$ consists of polynomials over variables in $\mathbf{r}$ and **abenc**, as well as lone variables $\hat{\mathbf{r}}$, which can also be determined in this pass (by checking for all monomials in each polynomial if it is a product of a known variable and a variable that does not occur in **abenc** - then it is a lone key variable).
- The ciphertext encodings **cenc** are split in $\mathbf{s}$ and $\overline{\mathbf{c}}$, where $\mathbf{s}$ consists of non-lone variables that occur only in the ciphertexts (and not in the master key encodings), and $\overline{\mathbf{c}}$ consists of polynomials over variables in $\mathbf{s}$ and **abenc**, as well as lone variables $\hat{\mathbf{s}}$ and $\tilde{\mathbf{s}}$ (to be determined later).

The second pass: distinguishing master-key from common variables. In the second pass over the inputs, the master-key/semi-common variables $\boldsymbol{\alpha\beta}$ are distinguished from the common variables $\mathbf{b}$. Note that we treat master-key/semi-common variables the same for the proof generation. We do this by going over the key polynomials:

- For each key polynomial k_i in $\mathbf{k}$, we inspect all monomials. If a monomial is
  - a product of a known variable and a variable α that occurs in **abenc**, then we put the variable α in $\boldsymbol{\alpha\beta}$;

  - a product of a known variable, a variable that occurs in $\mathbf{r}$ and a variable b that occurs in **abenc**, then we put the variable b in $\mathbf{b}$.

- If the lists $\boldsymbol{\alpha\beta}$ and $\mathbf{b}$ are not disjoint (i.e., some variable in **abenc** occurs in both lists), then the encodings are not well-formed and the checker outputs that the input scheme is not a PES-ISA.
- If the lists $\boldsymbol{\alpha\beta}$ and $\mathbf{b}$ together do not contain all variables in **abenc**, keep a list of unused **abenc** variables: $\mathbf{abenc}'$.

The third pass: distinguishing primed from non-primed ciphertext polynomials. In the third pass over the inputs, the ciphertext polynomials are split in the set of non-primed ciphertext polynomials $\mathbf{c}$ and primed ciphertext polynomials $\mathbf{c}'$. For each polynomial in $\overline{\mathbf{c}}$, we do the following:

- If the polynomial contains only monomials of the form:
  - a product of a known variable, a non-lone variable in $\mathbf{s}$ and a common variable in $\mathbf{b}$;
  - a product of a known variable, a non-lone variable in $\mathbf{s}$ and a variable b' in unused master public key $\mathbf{abenc}'$;
  - a product of a known variable and a lone variable $\hat{s}$ that does not occur in **abenc**;

  then it is placed in $\mathbf{c}$, and the lone variables $\hat{s}$ are placed in $\hat{\mathbf{s}}$. Furthermore, if the polynomial contains at least on monomial of the first case and monomials of the second form, with variable b', then b' is placed in $\mathbf{b}$.
- If the polynomial contains only monomials of the form:
  - a product of a known variable, a non-lone variable in $\mathbf{s}$ and a master-key/semi-common variable in $\boldsymbol{\alpha\beta}$;
  - a product of a known variable, a non-lone variable in $\mathbf{s}$ and a variable α' in the unused master public key $\mathbf{abenc}'$;
  - a product of a known variable and a lone variable $\tilde{s}$ that does not occur in **abenc**;

  then it is placed in $\mathbf{c}'$, and the lone variables $\tilde{s}$ are placed in $\tilde{\mathbf{s}}$. Furthermore, if the polynomial contains at least on monomial of the first case and monomials of the second form, with variable α', then α' is placed in $\boldsymbol{\alpha\beta}$.
- If the polynomial does not satisfy any of the two forms above, then the encodings are not well-formed and the checker outputs that the input scheme is not a PES-ISA.

The fourth pass: validating all encodings. In the fourth pass, all encodings are validated.

- The master public key encodings: $\boldsymbol{\alpha\beta}$ and $\mathbf{b}$ should be disjoint. We can consider outputting when some of the MPK encodings are not used (outputting $\mathbf{abenc}'$).
- The key encodings:
  - The lists $\mathbf{r}$ and $\hat{\mathbf{r}}$ should be disjoint;
  - For each key polynomial k_i in $\mathbf{k}$, it should be verified that each monomial in it is of the form:

  - ∗ a product of a known variable and a master-key/semi-common variable in $\boldsymbol{\alpha\beta}$;
  - ∗ a product of a known variable, a non-lone variable in $\mathbf{r}$ and a common variable in $\mathbf{b}$;
  - ∗ a product a known variable and a lone variable in $\hat{\mathbf{r}}$.

  If any of these does not hold, then the key encodings are not well-formed and the checker outputs the input scheme is not a PES-ISA.

- The ciphertext encodings:
  - The lists $\mathbf{s}$, $\hat{\mathbf{s}}$ and $\tilde{\mathbf{s}}$ should be disjoint;
  - For each ciphertext polynomial c_i in the list of non-primed ciphertext encodings, it should be verified that each monomial in it is of the form:
    - ∗ a product of a known variable, a common variable in $\mathbf{b}$ and a non-lone variable in $\mathbf{s}$;
    - ∗ a product a known variable and a lone variable in $\hat{\mathbf{s}}$.
  - For each ciphertext polynomial c_i' in the list of primed ciphertext encodings as well as the blinding/target value c_M, it should be verified that each monomial in it is of the form;
    - ∗ a product of a known variable, a master-key/semi-common variable in $\boldsymbol{\alpha\beta}$ and a non-lone variable in $\mathbf{s}$;
    - ∗ a product a known variable and a lone variable in $\tilde{\mathbf{s}}$.

    If any of these does not hold, then the ciphertext encodings are not well-formed an the checker outputs that the input scheme is not a PES-ISA.

A.2 Proof generation - finding a suitable kernel vector

We describe the algorithm for the SSSP proof generation by first finding a suitable kernel vector (as in Theorems 2 and 2).

The inputs. The proof generation takes as input:

- The master public keys consisting of
  - The master-key/semi-common variables $\boldsymbol{\alpha\beta}$
  - The common variables $\mathbf{b}$
- The key encodings consisting of
  - The key polynomials $\mathbf{k}$
  - The non-lone key variables $\mathbf{r}$
  - The lone key variables $\hat{\mathbf{r}}$
- The ciphertext encodings consisting of
  - The non-primed ciphertext polynomials $\mathbf{c}$
  - The primed ciphertext polynomials $\mathbf{c}'$
  - The non-lone ciphertext variables $\mathbf{s}$
  - The (non-primed) lone ciphertext variables $\hat{\mathbf{s}}$
  - The (primed) lone ciphertext variables $\tilde{\mathbf{s}}$
  - The blinding value c_M

Step one: setting up the $\mathbf{p_{enc}}$ vector. In the first step, we set up the $\mathbf{p_{enc}}$ vector that we decompose in the next step. To simplify the eventual decomposition of the vector $\mathbf{w}$, we store another vector $\mathbf{p_{enc}}'$ alongside it that keeps track of which (products of) encodings are stored in the entries of $\mathbf{p_{enc}}$.

- Compute $s_j k_i$ for all s_j in $\mathbf{s}$ and k_i in $\mathbf{k}$, and store it in $\mathbf{p_{enc}}$ (and store a tuple $(\text{non-lone ct}, j, \text{key\_poly}, i)$ in $\mathbf{p_{enc}}'$ in the corresponding entries)
- Compute $r_j c_i$ for all r_j in $\mathbf{r}$ and c_i in $\mathbf{c}$, and store it in $\mathbf{p_{enc}}$ (and store a tuple $(\text{non-lone key}, j, \text{ct\_poly}, i)$ in $\mathbf{p_{enc}}'$ in the corresponding entries)
- Add c'_i for all c'_i in $\mathbf{c'}$ to $\mathbf{p_{enc}}$ (and store a tuple $(\text{primed\_ct}, i)$ in $\mathbf{p_{enc}}'$ in the corresponding entries)

Step two: decomposing the $\mathbf{p_{enc}}$ vector. Then, we decompose the vector $\mathbf{p_{enc}}$ in the product $\mathbf{p_{enc}} = \mathbf{M} \cdot \mathbf{v}^\intercal$ as follows:

- Set up an empty matrix $\mathbf{M}$ and empty vector $\mathbf{v}$
- Add the following monomials to $\mathbf{v}$:
  - $\alpha_j s_{j'}$ for all α_j in $\boldsymbol{\alpha\beta}$ and $s_{j'}$ in $\mathbf{s}$
  - $\tilde{s}_j$ for all $\tilde{s}_j$ in $\tilde{\mathbf{s}}$
  - $r_j s_{j'} b_k$ for all r_j in $\mathbf{r}$, $s_{j'}$ in $\mathbf{s}$ and b_k in $\mathbf{b}$
  - $r_j \hat{s}_{j'}$ for all r_j in $\mathbf{r}$ and $\hat{s}_{j'}$ in $\hat{\mathbf{s}}$
  - $\hat{r}_j s_{j'}$ for all $\hat{r}_j$ in $\hat{\mathbf{r}}$ and $\hat{s}_{j'}$ in $\hat{\mathbf{s}}$
- We have $\mathbf{v} = \mathbf{v}_1 \| \mathbf{v}_2$, where $\mathbf{v}_1$ consists of the monomials in the first two cases above ($\alpha_j s_{j'}$ and $\tilde{s}_j$) and $\mathbf{v}_2$ consists of the monomials in the rest
- Then, for each polynomial p_i in $\mathbf{p_{enc}}$, we construct a vector $\mathbf{M}_i$ such that $\mathbf{M}_i \cdot \mathbf{v}^\intercal = p_i$, and $\mathbf{M}_i$ consists of known variables. This is done by considering each monomial in the polynomial, identifying its known variable and which entry of $\mathbf{v}$ corresponds with the monomial – this entry is the entry of $\mathbf{M}_i$ that gets assigned that known variable. (The rest is set to 0.) We append the matrix $\mathbf{M}$ with the row $\mathbf{M}_i$.
- Lastly, decompose the blinding value $c_M = \mathbf{tv} \cdot \mathbf{v}^\intercal$.

Step three: make "corrupted-variables" basis for kernel of M. The ultimate goal is to find a kernel vector $\mathbf{w}$ for $\mathbf{M}$ that is independent of the keys and that is not orthogonal to $\mathbf{tv}$, i.e., $\mathbf{tv} \cdot \mathbf{w}^\intercal \neq 0$. Furthermore, we require that each entry of $\mathbf{w}$ for which the corresponding entry in $\mathbf{v}$ is associated with a corrupted variable to be 0. The goal of step three is to create a basis for the kernel of $\mathbf{M}$, and then reduce it to a basis that sets the entries associated to corrupted variables to 0. To do this, we do the following.

- Compute the basis $\mathcal{V}$ for the kernel of $\mathbf{M}$
- Create a list of indices $\mathcal{I}$ of $\mathbf{v}$ such that $i \in \mathcal{I}$ if the i-th entry of $\mathbf{v}'$ is associated with a corrupted variable
- Create a set $\mathcal{V}'$ of truncated vectors in $\mathcal{V}$ with respect to $\mathcal{I}$, i.e., each vector in $\mathcal{V}'$ is a vector in $\mathcal{V}$ truncated to the entries with indices in $\mathcal{I}$
- Create a matrix $\mathbf{V}'$ where each column is a vector in $\mathcal{V}'$
- Create a matrix $\mathbf{V}$ where each column is a vector in $\mathcal{V}$

- Compute a basis $\mathcal{W}$ for the kernel of $\mathbf{V}'$
- Compute the set of vectors $\mathcal{W}_{\text{cor}} = \{\mathbf{V} \cdot \mathsf{w}^\intercal \mid \mathsf{w} \in \mathcal{W}\}$, which is a basis for $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I}[(\mathsf{v})_i = 0]\}$ (Lemma 4)

Step four: make the basis $\mathcal{W}_{\text{cor}}$ key independent. The goal is to transform the basis $\mathcal{W}_{\text{cor}}$ into a basis for the subspace of kernel vectors that are independent of the key. With independent of the key, we mean that the entries of the basis vectors associated with the entries in $\mathbf{v}_1$ do not contain any known variables associated with the key that do not occur in the ciphertext. To do this, we do the distinguish between two cases, when $\mathbf{c}'$ is not empty, and when $\mathbf{c}'$ is not empty. In Appendices B.2 and B.3, we prove that our algorithms below work.

When $\mathbf{c}'$ is not empty, we take the following steps.

- Extract a list of known variables $\mathbf{kv}_k$ that occurs in the key encodings
- Extract a list of known variables $\mathbf{kv}_c$ that occurs in the ciphertext encodings
- Create a list of known variables $\mathbf{kv}_{knc}$ that occurs in the key encodings but not in the ciphertext encodings
- Decompose the primed ciphertext polynomials $\mathbf{c}'$ in the product $\mathbf{c}' = \mathbf{M}_{c'} \cdot \mathbf{v}_{c'}^\intercal$
- Compute a basis $\mathcal{V}_{c'}$ for $\mathbf{M}_{c'}$
- Create a truncated basis $\mathcal{W}_{\text{trunc,cor}}$ by truncating the basis vectors in $\mathcal{W}_{\text{cor}}$ to consider only the entries associated with $\mathbf{v}_1$
- Write each vector in $\mathcal{W}_{\text{trunc,cor}}$ as a linear combination of the vectors in $\mathcal{V}_{c'}$
- Create a matrix $\mathbf{M}_{lc}$ of these linear combinations, such that each row corresponds to a vector in $\mathcal{W}_{\text{trunc,cor}}$
- Create a matrix $\mathbf{M}_{\text{rref}}$ by putting the matrix $\mathbf{M}_{lc}$ in reduced row echelon form and remove the all-zero rows
- Remove each row from $\mathbf{M}_{\text{rref}}$ that is associated with known variables in $\mathbf{kv}_{knc}$ (i.e., that contains at least one entry that is a function of at least one variable in $\mathbf{kv}_{knc}$)
- Initialize an empty list that will become the new basis $\mathcal{W}_{\text{cor,ki}}$
- For each row $\mathbf{M}_{\text{rref},i}$ of $\mathbf{M}_{\text{rref}}$:
  - find a linear combination $\mathbf{e}_i$ of the rows of $\mathbf{M}_{lc}$ that yields that row, i.e., $\mathbf{e}_i \cdot \mathbf{M}_{lc} = \mathbf{M}_{\text{rref},i}$
  - create a new basis vector $\mathsf{w}_{\text{cor,ki}}$ by rescaling the vectors in $\mathcal{W}_{\text{cor}}$ according to the linear combinations, i.e., if $\mathbf{W}_{\text{cor}}$ is the matrix in which each column is a basis vector in $\mathcal{W}_{\text{cor}}$, then $\mathsf{w}_{\text{cor,ki}} = \mathbf{W}_{\text{cor}} \cdot \mathbf{e}_i^\intercal$
  - add $\mathsf{w}_{\text{cor,ki}}$ to the list $\mathcal{W}_{\text{cor,ki}}$
- Output the vector basis $\mathcal{W}_{\text{cor,ki}}$ for the subspace of the kernel of $\mathbf{M}$ in which each entry associated with a corrupted variable is 0 and all entries associated with $\mathbf{v}_1$ are independent of the key

When $\mathbf{c}'$ is empty, we take the following steps.

- Extract a list of known variables $\mathbf{kv}_k$ that occurs in the key encodings
- Extract a list of known variables $\mathbf{kv}_c$ that occurs in the ciphertext encodings
- Create a list of known variables $\mathbf{kv}_{knc}$ that occurs in the key encodings but not in the ciphertext encodings
- Create a truncated basis $\mathcal{W}_{\text{trunc,cor}}$ by truncating the basis vectors in $\mathcal{W}_{\text{cor}}$ to consider only the entries associated with $\mathbf{v}_1$
- Create a matrix $\mathbf{M}_{\text{trunc,cor}}$, where each row is a vector in $\mathcal{W}_{\text{trunc,cor}}$
- Let $\mathbf{M}_{\text{rref}}$ be matrix $\mathbf{M}_{\text{trunc,cor}}$ in reduced row echelon form
- Remove each row from $\mathbf{M}_{\text{rref}}$ that is associated with known variables in $\mathbf{kv}_{knc}$ (i.e., that contains at least one entry that is a function of at least one variable in $\mathbf{kv}_{knc}$)
- Initialize an empty list that will become the new basis $\mathcal{W}_{\text{cor,ki}}$
- For each row $\mathbf{M}_{\text{rref},i}$ of $\mathbf{M}_{\text{rref}}$:
  - find a linear combination $\mathbf{e}_i$ of the rows of $\mathbf{M}_{\text{trunc,cor}}$ that yields that row, i.e., $\mathbf{e}_i \cdot \mathbf{M}_{\text{trunc,cor}} = \mathbf{M}_{\text{rref},i}$
  - create a new basis vector $\mathsf{w}_{\text{cor,ki}}$ by rescaling the vectors in $\mathcal{W}_{\text{cor}}$ according to the linear combinations, i.e., if $\mathbf{W}_{\text{cor}}$ is the matrix in which each column is a basis vector in $\mathcal{W}_{\text{cor}}$, then $\mathsf{w}_{\text{cor,ki}} = \mathbf{W}_{\text{cor}} \cdot \mathbf{e}_i^\intercal$
  - add $\mathsf{w}_{\text{cor,ki}}$ to the list $\mathcal{W}_{\text{cor,ki}}$
- Output the vector basis $\mathcal{W}_{\text{cor,ki}}$ for the subspace of the kernel of $\mathbf{M}$ in which each entry associated with a corrupted variable is 0 and all entries associated with $\mathbf{v}_1$ are independent of the key

Step five: find a basis vector that is not orthogonal to tv. The goal is to find a kernel vector $\mathbf{w}$ for $\mathbf{M}$ that is independent of the keys and that is not orthogonal to $\mathbf{tv}$, i.e., $\mathbf{tv} \cdot \mathbf{w}^\intercal \neq 0$. Furthermore, we require that each entry of $\mathbf{w}$ for which the corresponding entry in $\mathbf{v}_1$ is associated with a corrupted variable to be 0. To do this, output a vector $\mathbf{w}$ in $\mathcal{W}_{\text{cor,ki}}$ such that $\mathbf{tv} \cdot \mathbf{w}^\intercal \neq 0$. If it does not exist, then the scheme does not satisfy the SSSP (Proposition 5). Otherwise, one of the vectors in the set $\mathcal{W}_{\text{cor,ki}}$ should satisfy the requirement.

A.3 Proof generation - decomposing the kernel vector

We now describe the second half of the proof-generation algorithm for SSSP, by decomposing the appropriate kernel vectors. Roughly, we create a matrix decomposition for the ciphertext encodings $\mathbf{c}$ in a similar way as for $\mathbf{p_{enc}}$. Then, we first construct the substitution vectors for the ciphertext variables in $\mathbf{s}$ and $\hat{\mathbf{s}}$ and the substitution matrices for the common variables in $\mathbf{b}$, by considering the kernel of the decomposition for the ciphertexts. We then use the suitable kernel vector $\mathbf{w}$ to construct the substitution vectors for the rest of the variables. We describe the steps in more concrete detail below.

The inputs. The second step of the proof generation takes as *additional* input the vector $\mathbf{w}$ produced by the first step of the proof generation algorithm.

Step one: decomposing the ciphertext polynomials. We decompose $\mathbf{c}$ in $\mathbf{M}_c \cdot \mathbf{v}_c^\intercal$, such that $\mathbf{v}_c$ consists of all entries of the form $s_j b_k$ and $\hat{s}_j$. Compute a basis $\mathcal{V}_c$ for the kernel of $\mathbf{M}_c$.

Step two: constructing substitution vectors/matrices for s, ŝ and b. We then construct substitution vectors for the variables in $\mathbf{s}$, $\hat{\mathbf{s}}$ and $\mathbf{b}$ as follows. Let d_1 be the number of (non-lone) variables in $\mathbf{s}$ and d_2 be the number of vectors in $\mathcal{V}_c$.

- For each variable s_j in $\mathbf{s}$, let j denote the entry in $\mathbf{s}$ (e.g., the first entry in $\mathbf{s}$ has $j = 0$, the second $j = 1$), and set $\mathbf{s}_j = \mathbf{1}_j^{d_1}$ to be its substitution matrix (i.e., a vector in which the j-th entry is 1 and the rest is 0)
- For each b_k in $\mathbf{b}$ that is corruptable, we return the substitution vector $\mathbf{0}^{d_1 \times d_2}$
- For each b_k in $\mathbf{b}$ that is not corruptable, we compute the substitution matrix $\mathbf{B}_k$ as follows:
  - Determine the indices of the entries in $\mathbf{v}_c$ in which b_k occurs
  - Truncate all vectors in $\mathcal{V}_c$ to consider only the entries associated with those indices
  - Reorder the entries of the truncated vectors so that they correspond to $b_k\mathbf{s}$, i.e., the first entry is associated with $s_0 b_k$, the second with $s_1 b_k$, etc.
  - Create $\mathbf{B}_k$ by using the resulting truncated vectors as its column vectors, which is a $(d_1 \times d_2)$-matrix, because d_2 corresponds to the number of basis vectors (i.e., the column vectors) and d_1 corresponds to the length of $\mathbf{s}$
- For each $\hat{s}_j$ in $\hat{\mathbf{s}}$, we compute the substitution vector $\hat{\mathbf{s}}_j$ as follows:
  - Initialize an empty list $\hat{\mathbf{s}}_j$
  - Determine the index of the entry in $\mathbf{v}_c$ in which $\hat{s}_j$ occurs
  - For each vector in basis $\mathcal{V}_c$, select the associated entry with that index, and append it to $\hat{\mathbf{s}}_j$
- (Make sure that the i-th column of each substitution matrix $\mathbf{B}_k$ and the j-th entry of each substitution vectors $\hat{\mathbf{s}}_j$ corresponds to the i-th vector in $\mathcal{V}_c$)

Step three: constructing substitution vectors for $\alpha\beta$. We then construct substitution vectors for the variables in $\boldsymbol{\alpha\beta}$, for which we use the kernel vector $\mathbf{w}$ generated previously. In particular, for each α_j in $\boldsymbol{\alpha\beta}$, we compute the substitution vector $\mathbf{a}_j$ as follows:

- Initialize an empty list for $\mathbf{a}_j$
- We select the entries in $\mathbf{v}$ corresponding to $\alpha_j s_{j'}$ and determine the associated indices $\mathcal{I}$
- We order the indices in $\mathcal{I}$ so that the j'-th entry in $\mathcal{I}$ corresponds to $\alpha_j s_{j'}$, i.e., the first entry corresponds to $\alpha_j s_0$, the second to $\alpha_j s_1$, etc.
- For each $i \in \mathcal{I}$, we select the i-th entry of $\mathbf{w}$ and append it to $\mathbf{a}_j$

Step four: constructing substitution vectors for $\hat{\mathbf{r}}$. We construct the substitution vectors for the variables in $\hat{\mathbf{r}}$, for which we use the kernel vector $\mathbf{w}$ generated previously. For each variable $\hat{r}_j$ in $\hat{\mathbf{r}}$, we compute the substitution vector $\hat{\mathbf{r}}_j$ as follows:

- Initialize an empty list for $\hat{\mathbf{r}}_j$
- We select the entries in $\mathbf{v}$ corresponding to $\hat{r}_j s_{j'}$ and determine the associated indices $\mathcal{I}$
- We order the indices in $\mathcal{I}$ so that the j'-th entry in $\mathcal{I}$ corresponds to $\hat{r}_j s_{j'}$, i.e., the first entry corresponds to $\hat{r}_j s_0$, the second to $\hat{r}_j s_1$, etc.
- For each $i \in \mathcal{I}$, we select the i-th entry of $\mathbf{w}$ and append it to $\hat{\mathbf{r}}_j$

Step five: constructing substitution vectors for $\tilde{\mathbf{s}}$. We construct the substitution values for the variables in $\tilde{\mathbf{s}}$, for which we use the kernel vector $\mathbf{w}$ generated previously. For each variable $\tilde{s}_j$ in $\tilde{\mathbf{s}}$, we set the substitution value $\tilde{\mathbf{s}}_j$ to be the entry in $\mathbf{w}$ corresponding to the entry in $\mathbf{v}$ with $\tilde{s}_j$.

Step six: constructing substitution vectors for $\mathbf{r}$. Lastly, we construct the substitution vectors for the variables in $\mathbf{r}$, for which we use both the kernel basis $\mathcal{V}_c$ (for the matrix $\mathbf{M}_c$) and the kernel vector $\mathbf{w}$ (of the matrix $\mathbf{M}$). For each variable r_j in $\mathbf{r}$, we compute the substitution vector $\mathbf{r}_j$ as follows:

- We select the entries in $\mathbf{v}$ corresponding to the monomials in which r_j occurs and determine the associated indices $\mathcal{I}$
- We order the indices in $\mathcal{I}$ so that the j'-th entry in $\mathcal{I}$ corresponds to $r_j(\mathbf{v}_c)_{j'}$, i.e., the first entry corresponds to $r_j(\mathbf{v}_c)_0$, the second to $r_j(\mathbf{v}_c)_1$, etc.
- Initialize an empty list $\mathbf{w}_{r_j}$
- For each $i \in \mathcal{I}$, we select the i-th entry of $\mathbf{w}$ and append it to $\mathbf{w}_{r_j}$
- Create a matrix $\mathbf{V}_{c,r_j}$ by taking all the vectors in $\mathcal{V}_c$ as its first $|\mathcal{V}_c|$ column vectors and the last column vector is $-\mathbf{w}_{r_j}$
- Compute a basis for the kernel of $\mathbf{V}_{c,r_j}$
- Find a vector v in the basis such that the last entry is nonzero
- Divide all entries of v by the last entry and remove the last entry
- Output the resulting vector as $\mathbf{r}_j = \mathsf{v}$

A.4 Verifying the proofs

Lastly, we verify the substitution vectors and matrices output by the tool. The substitution vectors and matrices should satisfy four properties:

- Substituting the vectors and matrices in the polynomials yields all-zero vectors (except for the blinding value c_M)
- All corrupted master-key/semi-common variables in $\boldsymbol{\alpha\beta}$ are substituted by all-zero vectors
- All corrupted common variables in $\mathbf{b}$ are substituted by all-zero matrices
- All entries of the substitution vectors for the master-key/semi-common variables $\boldsymbol{\alpha\beta}$ should be independent of the known variables in the keys

These properties are verified in four steps.

Step one: verifying the substitutions. For the blinding value c_M, we verify that the substitutions yield a nonzero value. For the following polynomials, we verify that the substitutions yield all-zero vectors:

- The key polynomials $\mathbf{k}$
- The non-primed ciphertext polynomials $\mathbf{c}$
- The primed ciphertext polynomials $\mathbf{c}'$

Step two: verifying the corrupted master-key/semi-common variables. For each master-key/semi-common variable that is corrupted, we verify that the substitution vector is all-zero.

Step three: verifying the common variables. For each common variable that is corrupted, we verify that the substitution matrix is all-zero.

Step four: verifying the key independence. For each master-key/semi-common variable, we verify that none of the entries in the substitution vector depends on known variables that occur in the keys but not in the ciphertext.

B Linear algebra tools

For our tool, we use many of the linear algebra results of ACABELLA [dlPVA23]. We also add several new results that we need to find all vectors with certain properties in the kernel.

B.1 Constructing the basis $\mathcal{W}_{\text{cor}}$

In the first lemma, we show how to construct a basis $\mathcal{W}_{\text{cor}}$ for the subspace of the kernel of $\mathbf{M}$, in which all entries $\mathcal{I}$ associated with the corrupted variables are set to 0. Note that this lemma is identical to Lemma 7 in [dlPVA23] apart from some notational changes as well as removing parts that are not required for the proof.

Lemma 4. *Let $\mathbf{M}$ be a matrix, let $\mathcal{V} = \{\mathsf{v}_1, ..., \mathsf{v}_n\}$ be a basis for the kernel of $\mathbf{M}$, and let $\mathcal{I}$ be a set of indices. Let $\mathcal{V}' = \{\mathsf{v}'_1, ..., \mathsf{v}'_n\}$ be the set of vectors in $\mathcal{V}$ truncated to only consider the indices in $\mathcal{I}$, i.e., $\mathsf{v}'_j = (\{(\mathsf{v}_j)_i\}_{i\in\mathcal{I}})$. Let $\mathbf{V} = (\mathsf{v}_1^\intercal, ..., \mathsf{v}_n^\intercal)$ and $\mathbf{V}' = ((\mathsf{v}'_1)^\intercal, ..., (\mathsf{v}'_n)^\intercal)$ be the matrices associated with $\mathcal{V}$ and $\mathcal{V}'$. Consider then a vector basis $\mathcal{W}$ of $\text{Ker}(\mathbf{V}') = \{\mathsf{w} \mid \mathbf{V}' \cdot \mathsf{w}^\intercal = \mathbf{0}^\intercal\}$. Then, $\mathcal{W}_{\text{cor}} = \{\mathbf{V} \cdot \mathsf{w}^\intercal \mid \mathsf{w} \in \mathcal{W}\}$ is a basis for $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \;\; [(\mathsf{v})_i = 0]\}$.*

Proof. The proof of Lemma 7 in [dlPVA23] applies. □

B.2 Constructing the basis $\mathcal{W}_{\text{cor,ki}}$ (when $|\mathbf{c}'| \geq 1$)

To prove that step four of our algorithm in Appendix A.2 works (when $|\mathbf{c}'| \geq 1$), we prove the following results. First, we prove that the resulting set of vectors $\mathcal{W}_{\text{cor,ki}}$ provides a basis for a space that is a linear subspace of the space spanned by the basis vectors $\mathcal{W}_{\text{cor}}$. Then, we prove that the space spanned by $\mathcal{W}_{\text{cor,ki}}$ consists only of vectors that are key independent in the entries associated with $\mathbf{v}_1$. After this, we show that all vectors in the space spanned by $\mathcal{W}_{\text{cor}}$ that are key independent are also in the space spanned by $\mathcal{W}_{\text{cor,ki}}$. Lastly, we show that the resulting set of vectors is linearly independent, making it a basis for the space $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \;\; [(\mathsf{v})_i = 0] \wedge \forall j \in \mathcal{J} \;\; [(\mathsf{v})_j \text{ is key independent}]\}$.

The span of $\mathcal{W}_{\text{cor,ki}}$ is a subspace of the span of $\mathcal{W}_{\text{cor}}$.

Proposition 4. *Let $\mathcal{W}_{\text{cor}}$ be a basis for $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \;\; [(\mathsf{v})_i = 0]\}$. Then, the space spanned by $\mathcal{W}_{\text{cor,ki}}$, as generated in step four of our algorithm in Appendix A.2, is a linear subspace of $\mathcal{W}_{\text{cor}}$.*

We prove this in several steps via the following lemmas.

Lemma 5. *Let $\mathcal{V}$ be a basis for the kernel of $\mathbf{M}$, and let $\mathcal{V}_{c'}$ be a basis for $\text{Ker}(\mathbf{M}_{c'}) = \{\mathsf{v} \mid \mathbf{M}_{c'} \cdot \mathsf{v}^\intercal = \mathbf{0}^\intercal\}$, where $\mathbf{M}_{c'} \cdot \mathbf{v}_{c'}^\intercal$ is the matrix decomposition of the primed ciphertext encodings $\mathbf{c}'$. Let $\mathcal{V}'$ be the set of vectors in $\mathcal{V}$ truncated to the $\mathbf{v}_1$ part, i.e., if $\mathcal{J}$ is the set of indices such that $(\mathbf{v})_j$ is either of the form $\alpha_j s_{j'}$ or $\tilde{s}_j$, then $\mathcal{V}' = \{(\{(\mathsf{v})_j\}_{j\in\mathcal{J}}) \mid \mathsf{v} \in \mathcal{V}\}$. Then, $\text{span}(\mathcal{V}')$ is a subspace of $\text{span}(\mathcal{V}_{c'})$.*

Proof. The proof for this is similar as for the ciphertext polynomials $\mathbf{c}$, which we have also proven (via [dlPVA23]) in the proof of Theorem 2. □

Note that, because the span of $\mathcal{W}_{\text{cor}}$ is a subspace of the span of $\mathcal{V}$, we also have that the span of the truncated vectors of $\mathcal{W}_{\text{cor}}$ is in the span of $\mathcal{V}_{c'}$.

Corollary 3. *Let $\mathcal{W}_{\text{cor}}$ be a basis for $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \;\; [(\mathsf{v})_i = 0]\}$, and let $\mathcal{W}'_{\text{cor}}$ be the set of vectors in $\mathcal{W}_{\text{cor}}$ truncated to the $\mathbf{v}_1$ part. Then, the span of $\mathcal{W}'_{\text{cor}}$ is a subspace of the span of $\mathcal{V}_{c'}$.*

As a result of Lemma 5, we can write each vector in $\mathcal{V}' = \{\mathsf{v}'_1, ..., \mathsf{v}'_{n'}\}$ as a linear combination of the vectors in $\mathcal{V}_{c'} = \{\mathsf{v}_{c',1}, ..., \mathsf{v}_{c',n_{c'}}\}$. The goal is to find linear combinations of the vectors in $\mathcal{V}_{c'}$ that span the same space as $\mathcal{V}'$, of which we can easily determine a basis that spans a subspace of $\text{span}(\mathcal{V}')$ that consists only of key-independent vectors. We do this by constructing a matrix consisting of linear coefficients such that the i-th row corresponds to v'_i and the coefficients $\mathsf{c}_{i,j}$ are such that $\mathsf{v}'_i = \sum_j \mathsf{c}_{i,j}\mathsf{v}_{c',j}$. Then, we consider the reduced row echelon form of this matrix, whose row span is equivalent to the row span of the original matrix. Additionally, the reduced row echelon form will simplify determining a basis for the subspace of $\mathcal{V}'$ consisting of key-independent vectors.

Lemma 6. *Let $\mathcal{V}'$ and $\mathcal{V}_{c'}$ be any sets of vectors such that $\text{span}(\mathcal{V}')$ is a subspace of $\text{span}(\mathcal{V}_{c'})$. Let $\mathbf{M}_{lc}$ denote the matrix in which the i-th row of $\mathbf{M}_{lc}$ consists of coefficients $(\mathsf{c}_{i,1}, ..., \mathsf{c}_{i,n_c})$ such that $\mathsf{v}'_i = \sum_j \mathsf{c}_{i,j}\mathsf{v}_{c',j}$. Let $\mathbf{M}_{\text{rref}}$ denote the reduced row echelon form of $\mathbf{M}_{lc}$. Let $\mathbf{V}_{c'}$ be the matrix in which the i-th column vector is $\mathsf{v}_{c',i}$, and let $\mathsf{v}_{\text{rref},i} = \mathbf{V}_{c'} \cdot \mathbf{M}_{\text{rref},i}^\intercal$ be the vector that is obtained by taking the i-th row of $\mathbf{M}_{\text{rref}}$ as the coefficients for a linear combination of the vectors in $\mathcal{V}_{c'}$. Then, $\text{span}(\mathcal{V}_{\text{rref}}) = \text{span}(\mathcal{V}')$, where $\mathcal{V}_{\text{rref}} = \{\mathsf{v}_{\text{rref},1}, ..., \mathsf{v}_{\text{rref},n'}\}$.*

Proof. For simplicity, we also write each row i of $\mathbf{M}_{\text{rref}}$ as coefficients $\hat{\mathsf{c}}_{i,j}$, so that each $\mathsf{v}_{\text{rref},i} = \sum_j \hat{\mathsf{c}}_{i,j}\mathsf{v}_{c',j}$. Furthermore, because the row spaces of $\mathbf{M}_{\text{rref}}$ and $\mathbf{M}_{lc}$ are equivalent, we can write each row of $\mathbf{M}_{\text{rref}}$ as a linear combination of the rows of $\mathbf{M}_{lc}$, i.e., $(\hat{\mathsf{c}}_{i,1}, ..., \hat{\mathsf{c}}_{i,n_c}) = \sum_j \mathsf{e}_{i,j}(\mathsf{c}_{j,1}, ..., \mathsf{c}_{j,n_c})$, meaning that $\hat{\mathsf{c}}_{i,j} = \sum_k \mathsf{e}_{i,k}\mathsf{c}_{k,j}$. Therefore, each vector $\mathsf{v}_{\text{rref},i}$ is a linear combination of the vectors in $\mathcal{V}'$, i.e.,

$$\begin{aligned}\mathsf{v}_{\text{rref},i} &= \sum_j \hat{\mathsf{c}}_{i,j}\mathsf{v}_{c',j} = \sum_{j,k} \mathsf{e}_{i,k}\mathsf{c}_{k,j}\mathsf{v}_{c',j} \\ &= \sum_k \mathsf{e}_{i,k} \left(\sum_j \mathsf{c}_{k,j}\mathsf{v}_{c',j} \right) = \sum_k \mathsf{e}_{i,k}\mathsf{v}'_k.\end{aligned}$$

We use this to prove that any $\hat{\mathsf{v}} \in \text{span}(\mathcal{V}_{\text{rref}})$ is also in $\text{span}(\mathcal{V}')$, i.e.,

$$\hat{\mathsf{v}} = \sum_i d_i \mathsf{v}_{\text{rref},i} = \sum_{i,j} d_i \mathsf{e}_{i,j}\mathsf{v}'_j$$

$$= \sum_j \left(\sum_i d_i \mathsf{e}_{i,j} \right) \mathsf{v}'_j,$$

so, indeed, $\hat{\mathsf{v}} \in \text{span}(\mathcal{V}')$. Proving that any $\hat{\mathsf{v}} \in \text{span}(\mathcal{V}')$ is also in $\hat{\mathsf{v}} \in \text{span}(\mathcal{V}_{\text{rref}})$ is almost the same, i.e., we write each row of $\mathbf{M}_{lc}$ as a linear combination of the rows in $\mathbf{M}_{\text{rref}}$ and then write each vector v'_i as a linear combination of the vectors in $\mathcal{V}_{\text{rref}}$. □

By extension, any subset of $\mathcal{V}_{\text{rref}}$ spans a linear subspace of $\text{span}(\mathcal{V}')$. From this, it also follows that $\mathcal{W}_{\text{cor,ki}}$ spans a linear subspace of the span of $\mathcal{W}_{\text{cor}} = \{\mathsf{w}_{\text{cor},1}, ..., \mathsf{w}_{\text{cor},n'}\}$, as required by the proposition. In particular, this follows from noting that the vectors in $\mathcal{W}_{\text{cor,ki}}$ are constructed from the same linear combinations that yield $\mathcal{V}_{\text{rref}}$. That is, if $\mathsf{v}_{\text{rref},i} \in \mathcal{V}_{\text{rref}}$ is generated by computing $\mathsf{v}_{\text{rref},i} = \sum_j \mathsf{e}_j \mathsf{v}'_j$, where $\mathcal{V}'$ is the set of vectors in $\mathcal{W}_{\text{cor}}$ truncated to the $\mathbf{v}_1$ part, then each vector $\mathsf{w}_{\text{cor,rref},i} = \sum_j \mathsf{e}_j \mathsf{w}_{\text{cor},j}$ is generated as a linear combination of vectors in $\mathcal{W}_{\text{cor}}$. As such, $\mathcal{W}_{\text{cor,ki}} = \{\mathsf{w}_{\text{cor,rref},1}, ..., \mathsf{w}_{\text{cor,rref},n'}\}$ spans a linear subspace of $\text{span}(\mathcal{W}_{\text{cor}})$.

Any vector in $\text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$ is key independent. Equipped with $\mathbf{M}_{\text{rref}}$ and $\mathcal{V}_{\text{rref}}$, it is much easier to determine a basis for the subspace of $\mathcal{V}'$ that consists of all key-independent vectors. In particular, we will construct this basis by taking the subset of rows of $\mathbf{M}_{\text{rref}}$ (and associated vectors in $\mathcal{V}_{\text{rref}}$) that do not depend on the known variables that occur in the keys but not the ciphertexts. It follows rather easily that the span of the resulting set of vectors yields a space of key-independent vectors (provided that the coefficients for each linear combination are key independent). To express a span consisting of only linear combinations with key-independent coefficients, we use the notation span_{ki}.

Lemma 7. *Let $\mathcal{V}'$, $\mathbf{M}_{lc}$, $\mathbf{M}_{\text{rref}}$ and $\mathcal{V}_{\text{rref}}$ be as in Lemma 6, and let Var_{knc} denote the set of known variables that occurs in the key but not in the ciphertext. Let $\mathcal{I}'$ denote the set indices corresponding to the rows of $\mathbf{M}_{\text{rref}}$ in which none of the entries consists of known variables in Var_{knc} or all zeros, and construct a submatrix $\mathbf{M}_{\text{rref,ki}}$ of $\mathbf{M}_{\text{rref}}$ from those rows. Let $\mathcal{V}_{\text{rref,ki}} = \{\mathsf{v}_{\text{rref},i} \mid i \in \mathcal{I}\} \subseteq \mathcal{V}_{\text{rref}}$ consist of the associated vectors. Then, $\text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$ is a subspace of $\text{span}(\mathcal{V}')$ consisting of only key-independent vectors.*

Proof. That $\text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$ is a subspace of $\text{span}(\mathcal{V}')$ is trivial, because $\mathcal{V}_{\text{rref,ki}} \subseteq \mathcal{V}'$. Furthermore, all the vectors in $\text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$ are key independent in the sense that none of the entries of any vector in this span consist of values that are dependent on the variables in Var_{knc}. Let $\hat{\mathsf{v}} \in \text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$, i.e., $\hat{\mathsf{v}} = \sum_{i \in \mathcal{I}} d_i \mathsf{v}_{\text{rref},i}$. By definition of span_{ki}, the coefficients d_i do not depend on the variables in Var_{knc}. Hence, we only have to prove that each vector $\mathsf{v}_{\text{rref},i}$ for $i \in \mathcal{I}$ is key independent. To show this, we use that each $\mathsf{v}_{\text{rref},i}$ is a linear combination of vectors in $\mathcal{V}_{c'}$, which are by definition also key independent. These linear combinations use the coefficients provided by $\mathbf{M}_{\text{rref}}$. Because $\mathbf{M}_{\text{rref,ki}}$ considers only the rows of $\mathbf{M}_{\text{rref}}$ that are key independent, the coefficients are key independent as well. Therefore, $\hat{\mathsf{v}}$ is key independent. □

Any key-independent vector in $\text{span}(\mathcal{V}')$ is in $\text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$.

Lemma 8. *Let $\mathcal{V}'$, $\mathbf{M}_{lc}$, $\mathbf{M}_{\text{rref}}$, $\mathcal{V}_{\text{rref}}$, Var_{knc}, $\mathbf{M}_{\text{rref,ki}}$ and $\mathcal{V}_{\text{rref,ki}}$ be as in Lemma 7. Then, any key-independent vector in $\text{span}(\mathcal{V}')$ is also in $\text{span}_{\text{ki}}(\mathcal{V}_{\text{rref,ki}})$.*

Proof. Let $\hat{\mathsf{v}} \in \text{span}(\mathcal{V}')$. Because $\text{span}(\mathcal{V}_{\text{rref}}) = \text{span}(\mathcal{V}')$, this means that we can write $\hat{\mathsf{v}}$ as a linear combination of vectors in $\mathcal{V}_{\text{rref}}$. We show that, in particular, there can be no nonzero coefficients for the vectors in $\{\mathsf{v}_{\text{rref},i} \mid i \in [n'] \setminus \mathcal{I} \wedge \mathsf{v}_{\text{rref},i} \neq \mathbf{0}\}$. Suppose that this is not the case, i.e., let $d_1, ..., d_{n'}$ be coefficients such that $\hat{\mathsf{v}} = \sum_i d_i \mathsf{v}_{\text{rref},i}$ for which there exists at least one $i_{\text{special}} \in [n'] \setminus \mathcal{I}$ such that $d_{i_{\text{special}}} \neq 0$ and $\mathsf{v}_{\text{rref},i_{\text{special}}} \neq \mathbf{0}$. We know that $\mathsf{v}_{\text{rref},i} = \sum_j \hat{\mathsf{c}}_{i,j} \mathsf{v}_{c',j}$. Because the coefficients $\hat{\mathsf{c}}_{i,j}$ come from the matrix $\mathbf{M}_{\text{rref}}$, which

is in reduced row echelon form, we know that, if $\hat{\mathsf{c}}_{i,j} \neq 0$ for some j, then $\hat{\mathsf{c}}_{i',j} = 0$ for all $i' \neq i$. Hence, we can write $\hat{\mathsf{v}}$ as a linear combination of vectors in $\mathcal{V}_{c'}$:

$$\hat{\mathsf{v}} = \sum_i d_i \mathsf{v}_{\mathrm{rref},i} = \sum_{i,j} d_i \hat{\mathsf{c}}_{i,j} \mathsf{v}_{c',j},$$

so the coefficient for $\mathsf{v}_{c',j}$ is $\sum_i d_i \hat{\mathsf{c}}_{i,j}$. Let $\mathcal{J}$ denote the set of j for which $d_{i_{\mathrm{special}}} \hat{\mathsf{c}}_{i_{\mathrm{special}},j} \neq 0$. Note that, for all these $j \in \mathcal{J}$, we have that $\hat{\mathsf{c}}_{i,j} = 0$ for all $i \neq i_{\mathrm{special}}$.

Furthermore, because the i_{special}-th row is not key independent, there should be at least two nonzero entries $\hat{\mathsf{c}}_{i_{\mathrm{special}},j_1}, \hat{\mathsf{c}}_{i_{\mathrm{special}},j_2} \neq 0$, and there should be at least one entry that is key dependent. Note that, because the first nonzero entry in the row needs to be 1 (per the definition of the reduced row echelon form), we cannot scale the row to be key independent. To "cancel out" the key-dependent entry, we need to cancel it out with the other vectors in the linear combination $\sum_i d_i \mathsf{v}_{\mathrm{rref},i}$. Let $j_{\mathrm{special}} \in \mathcal{J}$ denote the key dependent entry, i.e., $\hat{\mathsf{c}}_{i_{\mathrm{special}},j_{\mathrm{special}}}$ is key dependent. Putting this all together, we have

$$\hat{\mathsf{v}} = d_{i_{\mathrm{special}}} \hat{\mathsf{c}}_{i_{\mathrm{special}},j_{\mathrm{special}}} \mathsf{v}_{c',j_{\mathrm{special}}} + d_{i_{\mathrm{special}}} \sum_{j \in \mathcal{J} \setminus \{j_{\mathrm{special}}\}} \hat{\mathsf{c}}_{i_{\mathrm{special}},j} \mathsf{v}_{c',j} + \sum_{\substack{i \neq i_{\mathrm{special}} \\ j \in [n_c] \setminus \mathcal{J}}} d_i \hat{\mathsf{c}}_{i,j} \mathsf{v}_{c',j}.$$

Note that the coefficient for $\mathsf{v}_{c',j_{\mathrm{special}}}$ is now $d_{i_{\mathrm{special}}} \hat{\mathsf{c}}_{i_{\mathrm{special}},j_{\mathrm{special}}}$, which is key dependent and therefore, each nonzero entry of $d_{i_{\mathrm{special}}} \hat{\mathsf{c}}_{i_{\mathrm{special}},j_{\mathrm{special}}} \mathsf{v}_{c',j_{\mathrm{special}}}$ is key dependent. To ensure that the key-dependent entries are canceled out, the whole vector needs to be canceled out by the other vectors. However, because the set $\mathcal{V}_{c'}$ is a basis, all vectors in it are linearly independent, so this cannot be done. Hence, $\hat{\mathsf{v}}$ cannot be key independent, and as a result, there can be no nonzero coefficients for the vectors in $\{\mathsf{v}_{\mathrm{rref},i} \mid i \in [n'] \setminus \mathcal{I} \wedge \mathsf{v}_{\mathrm{rref},i} \neq \mathbf{0}\}$ for any key-independent vector in $\mathrm{span}(\mathcal{V}')$.

To wrap up the proof, we note that we have now proven that each key-independent $\hat{\mathsf{v}} \in \mathrm{span}(\mathcal{V}')$ is in $\mathrm{span}(\mathcal{V}_{\mathrm{rref,ki}})$. It is relatively easy to see now that it is also in $\mathrm{span}_{\mathrm{ki}}(\mathcal{V}_{\mathrm{rref,ki}})$, because all the vectors in $\mathcal{V}_{\mathrm{rref,ki}}$ are key independent and they are linearly independent. Via a same argument as above, it must therefore be the case that any key-independent vector in $\mathrm{span}(\mathcal{V}_{\mathrm{rref,ki}})$ is also in $\mathrm{span}_{\mathrm{ki}}(\mathcal{V}_{\mathrm{rref,ki}})$. □

The set $\mathcal{W}_{\mathrm{cor,ki}}$ is linearly independent.

Lemma 9. *The set of vectors $\mathcal{W}_{\mathrm{cor,ki}}$ is linearly independent. It is therefore a basis for $\{\mathsf{v} \in \mathrm{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \;\; [(\mathsf{v})_i = 0] \wedge \forall j \in \mathcal{J} \;\; [(\mathsf{v})_j$ is key independent$]\}$.*

Proof. This follows directly from the fact that the set of truncated vectors $\mathcal{V}_{\mathrm{rref,ki}}$ (that truncates the vectors of $\mathcal{W}_{\mathrm{cor,ki}}$ to the $\mathbf{v}_1$-part) are linearly independent. These vectors are linearly independent owing to the linear independence of the nonzero rows of $\mathbf{M}_{\mathrm{rref}}$ as well as the linear independence of $\mathcal{V}_{c'}$. □

B.3 Constructing the basis $\mathcal{W}_{\mathrm{cor,ki}}$ (when $|\mathbf{c}'| = 0$)

We now prove that our algorithm in Appendix A.2 works (when $|\mathbf{c}'| = 0$).

Lemma 10. *Let $\mathcal{W}_{\mathrm{cor}}$ be a basis for $\{\mathsf{v} \in \mathrm{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \;\; [(\mathsf{v})_i = 0]\}$, and let $\mathcal{W}'_{\mathrm{cor}}$ be the set of vectors in $\mathcal{W}_{\mathrm{cor}}$ truncated to the $\mathbf{v}_1$ part, i.e., if $\mathcal{J}$ is the set of indices such that $(\mathbf{v})_j$ is either of the form $\alpha_j s_{j'}$ or $\tilde{s}_j$, then $\mathcal{W}' = \{(\{(\mathsf{v})_j\}_{j \in \mathcal{J}}) \mid \mathsf{v} \in \mathcal{W}_{\mathrm{cor}}\}$. Let $\mathbf{M}_{\mathrm{trunc,cor}}$ be the matrix in which each row is a vector in $\mathcal{W}'$, and let $\mathbf{M}_{\mathrm{trunc,cor,rref}}$ be the reduced row echelon form of $\mathbf{M}_{\mathrm{trunc,cor}}$. Let $\mathbf{M}_{\mathrm{trunc,cor,rref,ki}}$ be $\mathbf{M}_{\mathrm{trunc,cor,rref}}$ but with the rows removed that are key dependent or all-zero. Then, $\mathrm{span}_{\mathrm{ki}}(\mathbf{M}_{\mathrm{trunc,cor,rref,ki}})$ is a key independent and linearly independent subspace of $\mathrm{span}(\mathbf{M}_{\mathrm{trunc,cor}})$.*

Proof. This follows immediately from the fact that all the rows of $\mathbf{M}_{\text{trunc,cor,rref,ki}}$ are key independent, and that the reduced row echelon form spans the same space as $\mathbf{M}_{\text{trunc,cor}}$. By taking a submatrix of the reduced row echelon form, we obtain a (possibly strictly smaller) subspace. Furthermore, the nonzero rows of the reduced row echelon form are linearly independent. □

Lemma 11. *Let $\mathcal{W}'$ and $\mathbf{M}_{\text{trunc,cor,rref,ki}}$ be as in Lemma 10, and let v be a key-independent vector in the* $\text{span}(\mathcal{W}')$. *Then,* $\mathsf{v} \in \text{span}_{\text{ki}}(\mathbf{M}_{\text{trunc,cor,rref,ki}})$.

Proof. Let v be a key-independent vector in the $\text{span}(\mathcal{W}')$. Suppose that it is not in $\text{span}_{\text{ki}}(\mathbf{M}_{\text{trunc,cor,rref,ki}})$. Then there must be at least one key-dependent row i_{special} of $\mathbf{M}_{\text{trunc,cor,rref}}$ for which $\mathsf{v} = \sum_i \mathsf{c}_i \mathbf{M}_{\text{trunc,cor,rref},i}$ with $\mathsf{c}_{i_{\text{special}}} \neq 0$ and $\mathbf{M}_{\text{trunc,cor,rref},i}$ is key dependent. Because $\mathbf{M}_{\text{trunc,cor,rref}}$ is in the reduced row echelon form, the nonzero entries of the row cannot be canceled by the other rows. Furthermore, the first nonzero entry in the row is 1, meaning that another entry in that row is key dependent. Dividing the row by the key dependent entry will make the entry that is 1 key dependent. So, v cannot have any nonzero coefficients for the key dependent rows of $\mathbf{M}_{\text{trunc,cor,rref}}$. And thus, it must be in $\text{span}(\mathbf{M}_{\text{trunc,cor,rref,ki}})$. Furthermore, it must be a linear combination for which the coefficients are key independent, because multiplying a row with a key dependent scalar yields a key dependent vector (as it cannot be canceled out by the other rows). Hence, $\mathsf{v} \in \text{span}_{\text{ki}}(\mathbf{M}_{\text{trunc,cor,rref,ki}})$. □

Lemma 12. *Let $\mathcal{W}_{\text{cor,ki}}$ be constructed as in step four of the algorithm in Appendix A.2. The set of vectors $\mathcal{W}_{\text{cor,ki}}$ is a basis for* $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \quad [(\mathsf{v})_i = 0] \wedge \forall j \in \mathcal{J} \quad [(\mathsf{v})_j \text{ is key independent}]\}$.

Proof. This follows directly from the fact that the set of row vectors $\mathbf{M}_{\text{trunc,cor,rref,ki}}$ (that truncates the vectors of $\mathcal{W}_{\text{cor,ki}}$ to the $\mathbf{v}_1$-part) are linearly independent. These vectors are linearly independent owing to the linear independence of the nonzero rows of $\mathbf{M}_{\text{trunc,cor,rref}}$. □

B.4 Completeness of the algorithm

Putting together the results of the previous subsections, we formulate the following proposition. By proving this proposition, we prove that our algorithm in Appendix A.2 is complete in that it finds a suitable kernel vector if one exists.

Proposition 5. *Let $\mathbf{M}$ be a matrix and tv be a target vector that is not in the span of the rows of $\mathbf{M}$. Let $\mathcal{I}$ be a set of indices associated with corrupted variables, and let $\mathcal{J}$ be a set of indices associated with the $\mathbf{v}_1$ part of the matrix. Let $\mathcal{W}_{\text{cor,ki}}$ be a basis for* $\{\mathsf{v} \in \text{Ker}(\mathbf{M}) \mid \forall i \in \mathcal{I} \quad [(\mathsf{v})_i = 0] \wedge \forall j \in \mathcal{J} \quad [(\mathsf{v})_j \text{ is key independent}]\}$. *Let $\mathcal{W}' \subseteq \mathcal{W}_{\text{cor,ki}}$ be such that $\mathcal{W}' = \{\mathsf{v} \in \mathcal{W}_{\text{cor,ki}} \mid \mathsf{tv} \cdot \mathsf{v}^\intercal \neq 0\}$. Then, if some vector $\bar{\mathbf{w}} \in \text{Ker}(\mathbf{M})$ exists with $\mathsf{tv} \cdot \bar{\mathbf{w}}^\intercal \neq 0$ and $(\bar{\mathbf{w}})_i = 0$ for all $i \in \mathcal{I}$ and $(\bar{\mathbf{w}})_j$ is key independent for all $j \in \mathcal{J}$, then $\mathcal{W}' \neq \emptyset$.*

Proof. Let $\mathcal{W}_{\text{cor,ki}} = \{\mathsf{w}_1, ..., \mathsf{w}_n\}$ for some $n \in \mathbb{N}$, and let $\bar{\mathbf{w}} \in \text{Ker}(\mathbf{M})$ be some vector such that $\mathsf{tv} \cdot \bar{\mathbf{w}}^\intercal \neq 0$ and $(\bar{\mathbf{w}})_i = 0$ for all $i \in \mathcal{I}$ and $(\bar{\mathbf{w}})_j$ is key independent for all $j \in \mathcal{J}$. This means that $\bar{\mathbf{w}} \in \text{span}(\mathcal{W}_{\text{cor,ki}})$, and in particular, that some coefficients $\mathsf{c}_1, ..., \mathsf{c}_n$ exist such that $\bar{\mathbf{w}} = \sum_i \mathsf{c}_i \mathsf{w}_i$. If $\mathcal{W}' = \emptyset$, then $\mathbf{tv} \cdot \mathsf{w}_i^\intercal = 0$. By extension, it then also follows that $\mathbf{tv} \cdot \bar{\mathbf{w}}^\intercal = \mathbf{tv} \cdot (\sum_i \mathsf{c}_i \mathsf{w}_i)^\intercal = \sum_i \mathsf{c}_i \mathbf{tv} \cdot \mathsf{w}_i^\intercal = \sum_i 0 = 0$. This is a contradiction, so it must be that $\mathcal{W}' \neq 0$. □

C Proof of Theorem 3

Proof. To prove the theorem, we look at the $(\mathsf{s},\mathsf{s},\mathsf{s})$-CPA game first and we will see that when aux_y is not used to generate (non-)lone key variables, then the proof will not require y to be announced at the beginning of the game.

Let A be an adversary in the $(\mathsf{s},\mathsf{s},\mathsf{s})$-CPA game with scheme ABE-ISA such that the (d_1, d_2) SSSP holds for some d_1, d_2. Also let d_1' be the number of non-lone CT variables that are paired with semi-common variables. Then we show how to use A to construct an adversary B against the (d_1, d_1', d_2)-spDBDH assumption. To help keep track of the terms that can be programmed and which need to be canceled, we use colors. We use red for the terms that cannot be canceled, and blue for terms for which some of the terms can be programmed but not all. If no colors are used, the terms can be programmed using known values or terms from the assumption. Furthermore, because the terms "in the exponent" are substantially large, we will use implicit representation of group elements, e.g., $h^r = [r]_{\mathbb{H}}$.

Simulating the public keys and ciphertexts related to $\boldsymbol{\beta}$. A large part of our reduction is similar to the reduction in [Ven23], except for the simulation of the public keys instantiated from $\boldsymbol{\beta}$, the ciphertexts instantiated from $\mathbf{c}''$ and the non-lone ciphertext variables $\mathbf{s}$ that are combined with the $\boldsymbol{\beta}$ variables in $\mathbf{c}''$. In the reduction, we simulate these using the terms from the (d_1, d_1', d_2)-spDBDH assumption that are restricted to one group, i.e., $[\mathsf{x}\mathsf{c}_\mathsf{i}]_{\mathcal{D}(\mathsf{i})}$, $\left[\frac{\mathsf{y}\mathsf{z}}{\mathsf{c}_\mathsf{i}}\right]_{\mathcal{D}(\mathsf{i})}$ and $\left[\frac{\mathsf{x}\mathsf{y}\mathsf{z}\mathsf{c}_\mathsf{i}}{\mathsf{c}_{\mathsf{i}'}}\right]_{\mathcal{D}(\mathsf{i})}$ (for $\mathsf{i} \neq \mathsf{i}'$). More specifically, these non-lone ciphertext variables use $[\mathsf{x}\mathsf{c}_\mathsf{i}]_{\mathcal{D}(\mathsf{i})}$, the public-key variables use $\left[\frac{\mathsf{y}\mathsf{z}}{\mathsf{c}_\mathsf{i}}\right]_{\mathcal{D}(\mathsf{i})}$, and we use $\left[\frac{\mathsf{x}\mathsf{y}\mathsf{z}\mathsf{c}_\mathsf{i}}{\mathsf{c}_{\mathsf{i}'}}\right]_{\mathcal{D}(\mathsf{i})}$ to simulate the parts of the polynomial that we cannot cancel out using SSSP. To enable simulation, it is essential that the substitution vectors for the non-lone variables (resp. the semi-common variables) do not have overlap in the nonzero entries if they are instantiated in different groups. For example, if $\mathbf{s}_j$ and $\mathbf{s}_{j'}$ both have a nonzero value in the substitution vector in the first entry and they are instantiated in different groups, this is problematic. Fortunately, we can prove from relatively simple-to-check properties that this follows from our restrictions on the distributions. In particular, this follows if the non-lone variables s_j are substituted by standard basis vectors $\mathbf{1}_j^{d_1}$ (which have pairwise no overlap among nonzero entries). (Note that our algorithms implemented in Section 3.6 yields substitution vectors of this form and can generally be applied to argue that this property holds if SSSP holds. Specifically, by applying Theorem 2 twice: first convert the SSSP proof into a kernel vector and then decompose the vector using our algorithm to obtain an SSSP proof with the required properties.) Subsequently, the substitution vectors of the semi-common variables β_k have no overlap in nonzero entries if they do not occur together with the same non-lone variables or in the same polynomials as other products of non-lone and semi-common variables. This follows from the fact that we can set the $(j+1)$-th entry of β_k to 0 if $s_j\beta_k$ does not occur in any of the polynomials. In evaluating the polynomials with the substitutions, $\mathbf{s}_j\mathbf{b}_k^\intercal = (\mathbf{b}_k^\intercal)_{j+1}$, we select the $(j+1)$-th entry of the substitution vector of β_k. If it is never selected (which is the case when $s_j b_k$ does not occur anywhere), then its value does not matter for the correctness of the evaluations and can thus be 0.

Handling multiple challenge queries. We will prove security for multiple challenge queries via a standard hybrid argument. That is, we let $\mathbf{G}_0$ be the game where all challenge queries are answered with c_M and $\mathbf{G}_{n_c}$ be the game where all challenge queries are answered with a random element from the target group. To transition from game $\mathbf{G}_{i'-1}$ to $\mathbf{G}_{i'}$, we replace c_M of the i'-th challenge query with a random element from the target group. Each transition can be bounded by an adversary $B_{i'}$ against (d_1, d_1', d_2)-spDBDH. In the following, we will describe the simulation of $B_{i'}$. For better readability, we will write B

instead of $B_{i'}$ and x instead of $x_{i'}$.

Simulation. Adversary B gets as input the terms from the assumption (cf. Definition 20) as well as a challenge T. Let $\mathrm{EncB}, \mathrm{EncR}, \mathrm{EncS}$ be the three algorithms that generate the necessary substitutions for the variables in the encodings. Adversary B now runs A and simulates the $(\mathsf{s},\mathsf{s},\mathsf{s})$-$\mathsf{CPA}$ game as follows:

- **Initialization:** Using oracles InitMPK, InitChall, InitKey and InitCor, A commits to MPK indices $\mathcal{I}'$, challenge predicates $x_1, ..., x_{n_c} \in \mathcal{X}$, key predicates $y_1, ..., y_{n_k}$ and a set of indices $\mathcal{I}_{\mathrm{cor}}$ indicating corruptions. Adversary B runs Param_i for all $i \in \mathcal{I}'$ and obtains variables $(\boldsymbol{\alpha}_i, \mathbf{b}_i, \boldsymbol{\beta}_i)$ such that the combined vectors $\boldsymbol{\alpha}$, $\mathbf{b}$, $\boldsymbol{\beta}$ (as in the property well-formed parameters, Definition 17) are of length n_α, n_β, n_b, respectively. We define $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}$ to be the corrupted subsets of those variables. B then runs $(\mathbf{a}_1, ..., \mathbf{a}_{n_\alpha}, \mathbf{b}_1, ..., \mathbf{b}_{n_\beta}, \mathbf{B}_1, ..., \mathbf{B}_{n_b}) \leftarrow \mathrm{EncB}(x_{i'}, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b})$ to obtain the necessary substitutions which will be used to compute MPK_i when $i \in \mathcal{I}'$ is queried to MPK.

- **MPK oracle:** B can basically generate all MPK_i at the beginning of the simulation and then only output those requested by A. It does so by sampling $\bar{\alpha}_j$, $\bar{b}_k$ and $\bar{\beta}_j$ uniformly from $\mathbb{Z}_p$ and then it sets

$$\left\{ [\alpha_j]_{\mathbb{G}_T} = e(g,h)^{\bar{\alpha}_j} \cdot \prod_{\mathsf{i}\in[d_1]} e\left([\mathsf{y}]_{\mathbb{G}}, \left[\frac{\mathsf{z}}{\mathsf{c_i}}\right]_{\mathbb{H}} \right)^{(\mathbf{a}_j)_\mathsf{i}} \right\}_{j\in[n_\alpha]},$$

$$\left\{ [\beta_j]_{\mathfrak{D}(\beta_j)} = [\bar{\beta}_j]_{\mathfrak{D}(\beta_j)} \cdot \prod_{\mathsf{i}\in[d_1]} \left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]_{\mathfrak{D}(b_j)}^{(\mathbf{b}_j)_\mathsf{i}} \right\}_{j\in[n_\beta]},$$

$$\left\{ [b_k]_{\mathfrak{D}(b_k)} = [\bar{b}_k]_{\mathfrak{D}(b_k)} \cdot \prod_{\substack{\mathsf{i}\in[d_1]\\ \mathsf{j}\in[d_2]}} \left[\frac{\mathsf{z}}{\mathsf{c_i c'_j}}\right]_{\mathfrak{D}(b_k)}^{(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}} \right\}_{k\in[n_b]}.$$

  Note that when $a_j \in \mathfrak{a}_1$, $\beta_j \in \mathfrak{a}_2$ and $b_k \in \mathfrak{b}$, then the last term is 1 since $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{B}$ equal 0. Thus, B can also answer queries to Cor for those indices. Note that $[\beta_k]_{\mathfrak{D}(\beta_k)}$ can be simulated like this because of the restrictions on the distributions, as well as our argument that $\mathbf{s}_j$ can be represented by the standard basis vectors. These ensure that substitution vectors $\mathbf{b}_k$ for the semi-common variables are nonzero only in the entries associated with the s_j variables that they are paired with. Thus, these variables can be simulated from the restricted terms in the assumption.

- **KeyGen oracle:** When queried on (j, y_j, aux_y), where aux_y has not yet been queried, we aggregate all $y_{j'}$ with the same aux_y to y. If aux_y has already been queried, we have already computed the key and just return those parts that correspond to y_j. Also note that if aux_y is empty (or not used to derive (non-)lone variables), then y_j is already the aggregated key predicate y and we do not need to rely on adversary A to announce it.

  To compute SK_y, B generates $(\mathbf{r}_1, ..., \mathbf{r}_{m_1}, \hat{\mathbf{r}}_1, ..., \hat{\mathbf{r}}_{m_2}) \leftarrow \mathrm{EncR}(x, y)$ and $\bar{r}_j \in_R \mathbb{Z}_p$ for all $j \in [m_1]$ and programs part of the ciphertext as

$$\left\{ [r_j]_{\mathfrak{D}(r_j)} = [\bar{r}_j]_{\mathfrak{D}(r_j)} \cdot \prod_{\mathsf{j}\in[d_2]} [\mathsf{yc'_j}]_{\mathfrak{D}(r_j)}^{(\mathbf{r}_j)_\mathsf{j}} \right\}_{j\in[m_1]}, \quad \{[k_i]_{\mathfrak{D}(k_i)}\}_{i\in[m_3]},$$

and implicitly sets

$$\left\{\hat{r}_j = \check{r}_j + \sum_{\mathsf{i}\in[d_1]} \left((\hat{\mathbf{r}}_j)_\mathsf{i} \cdot \frac{\mathsf{yz}}{\mathsf{c_i}}\right)\right\}_{j\in[m_2]},$$

such that

$$[k_i]_{\mathfrak{D}(k_i)} = \left[\sum_{j\in[n_\alpha]} \delta_{1,i,j}\alpha_j + \sum_{j\in[n_\beta]} \delta_{2,i,j}\beta_j + \sum_{j\in[m_2]} \hat{\delta}_{i,j}\hat{r}_j + \sum_{\substack{j\in[m_1]\\k\in[n_b]}} \delta_{3,i,j,k} r_j b_k\right]_{\mathfrak{D}(k_i)}$$

is programmed implicitly as follows:

$$\left[\sum_{j\in[n_\alpha]} \delta_{1,i,j}\alpha_j\right]_{\mathfrak{D}(k_i)} = \prod_{j\in[n_\alpha]} \left([\delta_{1,i,j}\bar{\alpha}_j]_{\mathfrak{D}(k_i)} \cdot \prod_{\mathsf{i}\in[d_1]} \left[(\mathbf{a}_j)_\mathsf{i}\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{\delta_{1,i,j}}_{\mathfrak{D}(k_i)}\right),$$

$$\left[\sum_{j\in[n_\beta]} \delta_{2,i,j}\beta_j\right]_{\mathfrak{D}(k_i)} = \prod_{j\in[n_\beta]} \left([\delta_{2,i,j}\bar{\beta}_j]_{\mathfrak{D}(k_i)} \cdot \prod_{\mathsf{i}\in[d_1]} \left[(\mathbf{b}_j)_\mathsf{i}\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{\delta_{2,i,j}}_{\mathfrak{D}(k_i)}\right),$$

$$\left[\sum_{j\in[m_2]} \hat{\delta}_{i,j}\hat{r}_j\right]_{\mathfrak{D}(k_i)} = \prod_{j\in[m_2]} \left(\left[\hat{\delta}_{i,j}\check{r}_j\right]_{\mathfrak{D}(k_i)} \cdot \left(\prod_{\mathsf{i}\in[d_1]} \left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{(\hat{\mathbf{r}}_j)_\mathsf{i}}_{\mathfrak{D}(k_i)}\right)^{\hat{\delta}_{i,j}}\right),$$

$$\left[\sum_{\substack{j\in[m_1]\\k\in[n_b]}} \delta_{3,i,j,k} r_j b_k\right]_{\mathfrak{D}(k_i)} = \prod_{\substack{j\in[m_1]\\k\in[n_b]}} \left([\bar{r}_j b_k + r_j \bar{b}_k - \bar{r}_j\bar{b}_k]_{\mathfrak{D}(k_i)} \cdot \prod_{\substack{\mathsf{i}\in[d_1]\\\mathsf{j},\mathsf{j}'\in[d_2]}} \left[\frac{\mathsf{yzc}'_{\mathsf{j}'}}{\mathsf{c_i c}'_\mathsf{j}}\right]^{(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{r}_j)_{\mathsf{j}'}}_{\mathfrak{D}(k_i)}\right)^{\delta_{3,i,j,k}},$$

where $\check{r}_j \in_R \mathbb{Z}_p$ for all $j \in [m_2]$. All the terms, including those of the form $\left[\frac{\mathsf{yzc}'_{\mathsf{j}'}}{\mathsf{c_i c}'_\mathsf{j}}\right]^{(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{r}_j)_{\mathsf{j}'}}_{\mathfrak{D}(k_i)}$ with $\mathsf{j} \neq \mathsf{j}'$, can be programmed using earlier-generated values or terms from the assumption, except the terms with $\left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]_{\mathfrak{D}(k_i)}$, which are canceled because of the symbolic property (which ensures that k_i evaluates to $\mathbf{0}^{d_1}$ when all variables are substituted by their associated vectors and matrices):

$$\prod_{j\in[n_\alpha]}\prod_{\mathsf{i}\in[d_1]} \left[(\mathbf{a}_j)_\mathsf{i}\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{\delta_{1,i,j}}_{\mathfrak{D}(k_i)} \cdot \prod_{j\in[n_\beta]}\prod_{\mathsf{i}\in[d_1]} \left[(\mathbf{b}_j)_\mathsf{i}\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{\delta_{2,i,j}}_{\mathfrak{D}(k_i)} \cdot \prod_{j\in[m_2]} \left(\prod_{\mathsf{i}\in[d_1]} \left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{(\hat{\mathbf{r}}_j)_\mathsf{i}}_{\mathfrak{D}(k_i)}\right)^{\hat{\delta}_{i,j}}$$

$$\cdot \prod_{\substack{j\in[m_1]\\k\in[n_b]}} \left(\prod_{\substack{\mathsf{i}\in[d_1]\\\mathsf{j}\in[d_2]}} \left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{r}_j)_\mathsf{j}}_{\mathfrak{D}(k_i)}\right)^{\delta_{3,i,j,k}}$$

$$= \prod_{\mathsf{i}\in[d_1]} \left[\frac{\mathsf{yz}}{\mathsf{c_i}}\right]^{\mathsf{exp}}_{\mathfrak{D}(k_i)} = 1,$$

where $\mathsf{exp} = \sum_{j\in[n_\alpha]} \delta_{1,i,j}(\mathbf{a}_j)_\mathsf{i} + \sum_{j\in[n_\beta]} \delta_{2,i,j}(\mathbf{b}_j)_\mathsf{i} + \sum_{j\in[m_2]} \hat{\delta}_{i,j}(\hat{\mathbf{r}}_j)_\mathsf{i} + \sum_{j\in[m_1],k\in[n_b],\mathsf{j}\in[d_2]} \delta_{3,i,j,k}(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{r}_j)_\mathsf{j} = 0$.

- **Challenge oracle:** Subsequently, B creates a ciphertext for x and sends CT_x along with the challenge c_M to A. For this, it first runs $(\mathbf{s}_0, ..., \mathbf{s}_{w_1}, \hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_{w_2}, \tilde{\mathbf{s}}_1, ..., \tilde{\mathbf{s}}_{w_2'}) \leftarrow \mathrm{EncS}(x)$, and it sets (explicitly and implicitly, respectively)

$$\left\{ [s_j]_{\mathfrak{D}(s_j)} = [\bar{s}_j]_{\mathfrak{D}(s_j)} \cdot \prod_{\mathsf{i}\in[d_1]} [(\mathbf{s}_j)_\mathsf{i} \mathsf{x} \mathsf{c}_\mathsf{i}]_{\mathfrak{D}(s_j)} \right\}_{j\in\overline{[w_1]}}, \left\{ \hat{s}_j = \check{s}_j + \sum_{\mathsf{j}\in[d_2]} \left((\hat{\mathbf{s}}_j)_\mathsf{j} \cdot \frac{\mathsf{x}\mathsf{z}}{\mathsf{c}'_\mathsf{j}} \right) \right\}_{j\in[w_2]}$$

where $\bar{s}_j \in_R \mathbb{Z}_p$. As mentioned earlier, $[s_j]_{\mathfrak{D}(s_j)}$ can be simulated like this because the restrictions on the distributions, as well as our argument that $\mathbf{s}_j$ can be represented by the standard basis vectors. These ensure that the non-lone variables that occur together with semi-common can be simulated from the restricted terms in the assumption. For all $i \in [w_3]$, it programs

$$[c_i]_{\mathfrak{D}(c_i)} = \left[\sum_{j\in[w_2]} \eta_{1,i,j} \hat{s}_j + \sum_{\substack{j\in\overline{[w_1]} \\ k\in[n_b]}} \eta_{1,i,j,k} s_j b_k \right]_{\mathfrak{D}(c_i)}$$

as follows:

$$\left[\sum_{j\in[w_2]} \eta_{1,i,j} \hat{s}_j \right]_{\mathfrak{D}(c_i)} = \prod_{j\in[w_2]} \left([\eta_{1,i,j} \check{s}_j]_{\mathfrak{D}(c_i)} \cdot \left(\prod_{\mathsf{j}\in[d_2]} \left[\frac{\mathsf{x}\mathsf{z}}{\mathsf{c}'_\mathsf{j}} \right]_{\mathfrak{D}(c_i)}^{(\hat{\mathbf{s}}_j)_\mathsf{j}} \right)^{\eta_{1,i,j}} \right)$$

$$\left[\sum_{\substack{j\in\overline{[w_1]} \\ k\in[n_b]}} \eta_{1,i,j,k} s_j b_k \right]_{\mathfrak{D}(c_i)} = \prod_{\substack{j\in[w_1] \\ k\in[n_b]}} \left([\bar{s}_j b_k + s_j \bar{b}_k - \bar{s}_j \bar{b}_k]_{\mathfrak{D}(c_i)} \cdot \prod_{\substack{\mathsf{i},\mathsf{i}'\in[d_1] \\ \mathsf{j}\in[d_2]}} \left[\frac{\mathsf{x}\mathsf{z}\mathsf{c}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}\mathsf{c}'_\mathsf{j}} \right]_{\mathfrak{D}(c_i)}^{(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{s}_j)_{\mathsf{i}'}} \right)^{\eta_{1,i,j,k}}$$

where $\check{s}_j \in_R \mathbb{Z}_p$ for all $j \in [w_2]$. Furthermore, all the terms, including those of the form $\left[\frac{\mathsf{x}\mathsf{z}\mathsf{c}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}\mathsf{c}'_\mathsf{j}}\right]_\mathbb{G}$ with $\mathsf{i} \neq \mathsf{i}'$, can be programmed using earlier-generated values or terms from the assumption, except the terms with $\left[\frac{\mathsf{x}\mathsf{z}}{\mathsf{c}'_\mathsf{j}}\right]_\mathbb{G}$, which are canceled because of the symbolic property (which ensures that c_i evaluates to $\mathbf{0}^{d_2}$ when all variables are substituted by their associated vectors and matrices):

$$\prod_{j\in[w_2]} \prod_{\mathsf{j}\in[d_2]} \left[\frac{\mathsf{x}\mathsf{z}}{\mathsf{c}'_\mathsf{j}} \right]_{\mathfrak{D}(c_i)}^{\eta_{1,i,j}(\hat{\mathbf{s}}_j)_\mathsf{j}} \cdot \prod_{\substack{j\in[w_1] \\ k\in[n_b] \\ \mathsf{i}\in[d_1] \\ \mathsf{j}\in[d_2]}} \left[\frac{\mathsf{x}\mathsf{z}}{\mathsf{c}'_\mathsf{j}} \right]_{\mathfrak{D}(c_i)}^{\eta_{1,i,j,k}(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{s}_j)_{\mathsf{i}'}} = \prod_{\mathsf{j}\in[d_2]} \left[\frac{\mathsf{x}\mathsf{z}}{\mathsf{c}'_\mathsf{j}} \right]_{\mathfrak{D}(c_i)}^{\mathsf{exp}} = 1,$$

where $\mathsf{exp} = \sum_{j\in[w_2]} \eta_{1,i,j}(\hat{\mathbf{s}}_j)_\mathsf{j} + \sum_{j\in[w_1],k\in[n_b],\mathsf{i}\in[d_1]} \eta_{1,i,j,k}(\mathbf{B}_k)_{\mathsf{i},\mathsf{j}}(\mathbf{s}_j)_\mathsf{i} = 0$.
Furthermore, it programs

$$[c'_i]_{\mathbb{G}_T} = \left[\sum_{j\in[n_\alpha],j'\in\overline{[w_1]}} \eta'_{1,i,j,j'} \alpha_j s_{j'} + \sum_{j\in[n_\beta],j'\in\overline{[w_1]}} \eta'_{2,i,j,j'} \beta_j s_{j'} + \sum_{j\in[w_2']} \hat{\eta}'_{i,j} \tilde{s}_j \right]_{\mathbb{G}_T}$$

for all $i \in [w_4]$ as follows:

$$\left[\sum_{\substack{j\in[n_\alpha] \\ j'\in\overline{[w_1]}}} \eta'_{1,i,j,j'} \alpha_j s_{j'} \right]_{\mathbb{G}_T} = \prod_{\substack{j\in[n_\alpha] \\ j'\in\overline{[w_1]}}} \left([\bar{\alpha}_j s_{j'} + \alpha_j \bar{s}_{j'} - \bar{\alpha}_j \bar{s}_{j'}]_{\mathbb{G}_T} \cdot \prod_{\mathsf{i},\mathsf{i}'\in[d_1]} \left[\frac{\mathsf{x}\mathsf{y}\mathsf{z}\mathsf{c}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}} \right]_{\mathbb{G}_T}^{(\mathbf{a}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'}} \right)^{\eta'_{1,i,j,j'}}$$

$$\left[\sum_{\substack{j\in[n_\beta]\\ j'\in\overline{[w_1]}}} \eta'_{2,i,j,j'}\beta_j s_{j'}\right]_{\mathbb{G}_T} = \prod_{\substack{j\in[n_\alpha]\\ j'\in\overline{[w_1]}}} \left(\left[\bar{\beta}_j s_{j'} + \beta_j \bar{s}_{j'} - \bar{\beta}_j \bar{s}_{j'}\right]_{\mathbb{G}_T} \cdot \prod_{\mathsf{i},\mathsf{i}'\in[d_1]} \left[\frac{\mathsf{xyzc}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}}\right]_{\mathbb{G}_T}^{(\mathbf{b}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'}} \right)^{\eta'_{2,i,j,j'}}$$

$$\left[\sum_{j\in[w'_2]} \hat{\eta}'_{i,j}\tilde{s}_j\right]_{\mathbb{G}_T} = \prod_{j\in[w'_2]} \left([\breve{s}_j]_{\mathbb{G}_T} \cdot [\mathsf{xyz}]_{\mathbb{G}_T}^{\tilde{\mathbf{s}}_j} \right)^{\hat{\eta}'_{i,j}},$$

where all the terms, including those with

$$\left[\frac{\mathsf{xyzc}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}}\right]_{\mathbb{G}_T} = e\left([\mathsf{xc}_\mathsf{i}]_\mathbb{G}, \left[\frac{\mathsf{yzc}'_\mathsf{j}}{\mathsf{c}_\mathsf{i}\mathsf{c}'_{\mathsf{j}'}}\right]_\mathbb{H}\right)$$

for $\mathsf{i} \neq \mathsf{i}'$, can be programmed using earlier-generated values or terms from the assumption, except the terms with $[\mathsf{xyz}]_{\mathbb{G}_\mathsf{T}}$, which are canceled because of the symbolic property (which ensures that c'_i evaluates to 0 when all variables are substituted by their associated vectors and values):

$$\prod_{\substack{j\in[n_\alpha]\\ j'\in\overline{[w_1]}\\ \mathsf{i}\in[d_1]}} [\mathsf{xyz}]_{\mathbb{G}_T}^{\eta'_{1,i,j,j'}(\mathbf{a}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'}} \cdot \prod_{\substack{j\in[n_\beta]\\ j'\in\overline{[w_1]}\\ \mathsf{i}\in[d_1]}} [\mathsf{xyz}]_{\mathbb{G}_T}^{\eta'_{2,i,j,j'}(\mathbf{b}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'}} \cdot \prod_{j\in[w'_2]} [\mathsf{xyz}]_{\mathbb{G}_T}^{\hat{\eta}'_{i,j}\tilde{\mathbf{s}}_j} = [\mathsf{xyz}]_{\mathbb{G}_T}^{\mathsf{exp}} = 1,$$

where $\mathsf{exp} = \sum_{j\in[n_\alpha],j'\in\overline{[w_1]},\mathsf{i}\in[d_1]} \eta'_{1,i,j,j'}(\mathbf{a}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'} + \sum_{j\in[n_\beta],j'\in\overline{[w_1]},\mathsf{i}\in[d_1]} \eta'_{2,i,j,j'}(\mathbf{b}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'} + \sum_{j\in[w'_2]} \hat{\eta}'_{i,j}\tilde{\mathbf{s}}_j = 0$.

Ciphertext elements $\mathbf{c}''$ can be simulated similar to $\mathbf{c}$. Adversary B programs

$$[c''_i]_{\mathfrak{D}(c''_i)} = \left[\sum_{j\in[w'_2]} \hat{\eta}_{2,i,j}\tilde{s}_j + \sum_{\substack{j\in\overline{[w_1]}\\ k\in[n_\beta]}} \eta_{2,i,j,k}s_j\beta_k\right]_{\mathfrak{D}(c''_i)}$$

as follows:

$$\left[\sum_{j\in[w'_2]} \hat{\eta}_{2,i,j}\tilde{s}_j\right]_{\mathfrak{D}(c''_i)} = \prod_{j\in[w'_2]} \left([\hat{\eta}_{2,i,j}\breve{s}_j]_{\mathfrak{D}(c''_i)} \cdot \left(\prod_{\mathsf{j}\in[d_2]} [\mathsf{xyz}]_{\mathfrak{D}(c''_i)}^{(\tilde{\mathbf{s}}_j)_\mathsf{j}} \right)^{\hat{\eta}_{2,i,j}} \right)$$

$$\left[\sum_{\substack{j\in\overline{[w_1]}\\ k\in[n_b]}} \eta_{1,i,j,k}s_j\beta_k\right]_{\mathfrak{D}(c''_i)} = \prod_{\substack{j\in[w_1]\\ k\in[n_\beta]}} \left(\left[\bar{s}_j\beta_k + s_j\bar{\beta}_k - \bar{s}_j\bar{\beta}_k\right]_{\mathfrak{D}(c''_i)} \cdot \prod_{\mathsf{i},\mathsf{i}'\in[d_1]} \left[\frac{\mathsf{xyzc}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}}\right]_{\mathfrak{D}(c''_i)}^{(\mathbf{b}_k)_\mathsf{i}(\mathbf{s}_j)_{\mathsf{i}'}} \right)^{\eta_{2,i,j,k}}$$

where all the terms, including those of the form $\left[\frac{\mathsf{xyzc}_{\mathsf{i}'}}{\mathsf{c}_\mathsf{i}}\right]_{\mathfrak{D}(c''_i)}$ with $\mathsf{i} \neq \mathsf{i}'$, can be programmed using earlier-generated values or terms from the assumption, except the terms with $[\mathsf{xyz}]_{\mathfrak{D}(c''_i)}$, which are canceled because of the symbolic property (which ensures that c''_i evaluates to 0 when all variables are substituted by their associated vectors and matrices):

$$\prod_{\substack{j\in\overline{[w_1]}\\ k\in[n_\beta]}} [\mathsf{xyz}]_{\mathfrak{D}(c''_i)}^{\eta_{2,i,j,k}\mathbf{s}_j\mathbf{b}_k} \cdot \prod_{j\in[w'_2]} [\mathsf{xyz}]_{\mathfrak{D}(c''_i)}^{\hat{\eta}_{2,i,j}\tilde{\mathbf{s}}_j} = [\mathsf{xyz}]_{\mathfrak{D}(c''_i)}^{\mathsf{exp}} = 1,$$

where $\mathsf{exp} = \sum_{j\in\overline{[w_1]},k\in[n_\beta]} \eta_{2,i,j,k}\mathbf{s}_j\mathbf{b}_k + \sum_{j\in[w_2']} \hat{\eta}_{2,i,j}\tilde{\mathbf{s}}_j = 0$.

Note that the terms of the form $\left[\frac{\mathsf{xyzc}_{i'}}{\mathsf{c}_i}\right]_{\mathfrak{D}(c_i'')}$ can be simulated because we had shown that the substitution vectors for β_k are nonzero in only the entries associated with the non-lone variables that they occur together with. In accordance with our restrictions on the distributions, these should occur in the same groups, as well as their associated polynomials. Thus, all the nonzero products are also in the same group. Lastly, the challenge is computed as follows:

$$
\begin{aligned}
e(g,h)^{c_M} &= \left[\sum_{j\in[w_2']} \zeta_j\tilde{s}_j + \sum_{\substack{j\in[n_\alpha]\\ j'\in\overline{[w_1]}}} \zeta_{1,j,j'}\alpha_j s_{j'} + \sum_{\substack{j\in[n_\beta]\\ j'\in\overline{[w_1]}}} \zeta_{2,j,j'}\beta_j s_{j'}\right]_{\mathbb{G}_T} \\
&= \prod_{j\in[w_2']} \left([\breve{s}_j]_{\mathbb{G}_T} \cdot [\mathsf{xyz}]_{\mathbb{G}_T}^{\tilde{\mathbf{s}}_j}\right)^{\zeta_j} \\
&\quad \cdot \prod_{\substack{j\in[n_\alpha]\\ j'\in\overline{[w_1]}}} \left([\bar{\alpha}_j s_{j'} + \alpha_j\bar{s}_{j'} - \bar{\alpha}_j\bar{s}_{j'}]_{\mathbb{G}_T} \cdot \prod_{\mathsf{i},\mathsf{i}'\in[d_1]} \left[\frac{\mathsf{xyzc}_{i'}}{\mathsf{c}_i}\right]_{\mathbb{G}_T}^{(\mathbf{a}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'}}\right)^{\zeta_{1,j,j'}} \\
&\quad \cdot \prod_{\substack{j\in[n_\beta]\\ j'\in\overline{[w_1]}}} \left([\bar{\beta}_j s_{j'} + \beta_j\bar{s}_{j'} - \bar{\beta}_j\bar{s}_{j'}]_{\mathbb{G}_T} \cdot \prod_{\mathsf{i},\mathsf{i}'\in[d_1]} \left[\frac{\mathsf{xyzc}_{i'}}{\mathsf{c}_i}\right]_{\mathbb{G}_T}^{(\mathbf{b}_j)_\mathsf{i}(\mathbf{s}_{j'})_{\mathsf{i}'}}\right)^{\zeta_{2,j,j'}}
\end{aligned}
$$

where, again, most of the terms can be programmed using earlier-generated values or terms from the assumption, except the following part, which is replaced by the challenge value T from the (d_1, d_1', d_2)-spDBDH assumption:

$$
\prod_{j\in[w_2']} [\mathsf{xyz}]_{\mathbb{G}_T}^{\zeta_j\tilde{\mathbf{s}}_j} \cdot \prod_{\substack{j\in[n_\alpha]\\ j'\in\overline{[w_1]}\\ \mathsf{i}\in[d_1]}} [\mathsf{xyz}]_{\mathbb{G}_T}^{\zeta_{1,j,j'}(\mathbf{a}_j)_\mathsf{i}(\mathbf{s}_{j'})_\mathsf{i}} \cdot \prod_{\substack{j\in[n_\beta]\\ j'\in\overline{[w_1]}\\ \mathsf{i}\in[d_1]}} [\mathsf{xyz}]_{\mathbb{G}_T}^{\zeta_{2,j,j'}(\mathbf{b}_j)_\mathsf{i}(\mathbf{s}_{j'})_\mathsf{i}} = T^\Gamma,
$$

where $\Gamma = \sum_{j\in[w_2']} \zeta_j\tilde{\mathbf{s}}_j + \sum_{j\in[n_\alpha],j'\in\overline{[w_1]},\mathsf{i}\in[d_1]} (\mathbf{a}_j)_\mathsf{i}(\mathbf{s}_{j'})_\mathsf{i} + \sum_{j\in[n_\beta],j'\in\overline{[w_1]},\mathsf{i}\in[d_1]} (\mathbf{b}_j)_\mathsf{i}(\mathbf{s}_{j'})_\mathsf{i} \neq 0$ because of the symbolic property. Note that this ciphertext component is well-formed if $T = [\mathsf{xyz}]_{\mathbb{G}_T}$, and uniformly random if not.

- **Random oracle queries:** If aux_y is used to generate non-lone key variables via the random oracle, then the relevant queries are known to the challenger at the start of the game and can be simulated accordingly. Other outputs of the form $[b_k]$ or $[s_j]$ can also be simulated as described above. When asked for inputs that are not related to the predicates, B simply draws a random group element.

- **Decision phase:** Adversary A outputs a guess δ' and adversary B outputs the same bit to its own challenger.

All remaining challenge ciphertexts are simulated as described in the hybrid game. Note that whenever A distinguishes between hybrid $\mathbf{G}_{i'-1}$ and $\mathbf{G}_{i'}$, then adversary B solves (d_1, d_1', d_2)-spDBDH. Hence, collecting the probabilities over all hybrids yields the claimed bound. □

D The generic group model and omitted proofs

In the generic group model (GGM), the adversary gets access to the (pairing) group operations via oracles. Each group element is represented by a unique handle which is usually a random string or an index in a list of already seen elements. In this work, we will consider the latter approach which is essentially based on the model by Maurer [Mau05]. The group operation oracle Op takes as input two handles to group elements g^a and g^b and outputs a handle to $g^c = g^{a+b}$. In the pairing group setting, the oracle additionally takes an identifier indicating whether the operation is performed in group $\mathbb{G}$, $\mathbb{H}$ or $\mathbb{G}_T$. The pairing oracle Pair takes as input handles to group elements g^a and h^b and outputs a handle to g_T^{ab}. When analyzing a complexity assumption or scheme in the GGM, we only look at the exponents and simulate via polynomials. This is also sometimes referred to as the symbolic model. We argue that a certain polynomial cannot be created using those polynomials that the adversary knows, showing that the adversary cannot win in the symbolic model. It then remains to show that the simulation using polynomials is statistically close to the simulation in the GGM. We recall a useful lemma that is often used in those proofs.

Lemma 13 (Schwartz-Zippel Lemma)**.** *Let $f(x_1, .., x_n)$ be a non-zero multivariate polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$. Let $\alpha_1, \ldots, \alpha_n$ be chosen uniformly at random from S. Then*

$$\Pr[f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|} .$$

D.1 Generic hardness of (d_1, d_1', d_2)-spDBDH

We prove GGM bounds for our q-type assumption given in Definition 20.

Theorem 5. *Let d_1, d_1', d_2 be positive integers such that $d_1' \leq d_1$. Let A be a generic adversary against the (d_1, d_1', d_2)-spDBDH assumption, issuing at most n_{op} queries to Op and n_{pair} queries to Pair. Then,*

$$\mathsf{Adv}^{(d_1,d_1',d_2)\text{-spDBDH}}_{\mathsf{PGGen},A}(\lambda) \leq \frac{(2d_1d_2 + 6) \cdot n^2}{p} ,$$

where $n = (n_{\text{op}} + n_{\text{pair}} + 3 + 3\max\{d_1^2 d_2, d_1 d_2^2\})$.

Proof. We want to simulate the terms provided by the assumption via polynomials. At the beginning of the game, the challenger initializes lists $\mathcal{L}_{\mathbb{G}}$ and $\mathcal{L}_{\mathbb{H}}$ with the generators, i.e., $\mathcal{L}_{\mathbb{G}}[1] = \mathcal{L}_{\mathbb{H}}[1] = 1$ as well as the terms provided by the assumption. List $\mathcal{L}_{\mathbb{G}_T}$ is initialized with the challenge, that is either the polynomial xyz or an independent formal variable u. The challenger outputs the indices to the adversary who can now perform operations via oracles Op and Pair. This will add new polynomials to the lists. At the end, the adversary will output a bit. We want to show that when the simulator uses polynomials the adversary can only win with probability 1/2. Further, we analyze the probability that the simulation fails which happens if there exist two polynomials p and p' such that the polynomials are not equal, but they evaluate to the same value when assigning random values to the formal variables.

Bounding the degree of polynomials. Since some of the exponents are rational fractions, the degree of polynomials may increase when the adversary performs group operations via oracle Op. To bound the maximum degree of polynomials the adversary creates (in each of the groups), we view an exponent ρ as the ratio of two polynomials p_1 and p_2 and compute the degree of p_1/p_2 as $\max\{\deg(p_1), \deg(p_2)\}$.

We observe that the least common multiple of all denominators of terms is $\Delta = \prod_{\mathsf{i}\in[d_1],\mathsf{j}\in[d_2]} \mathsf{c_i c'_j}$. That is, the adversary can create polynomials of degree at most $\delta_{\mathbb{G}'} = \max_i\{\deg(\Delta \cdot \rho_{\mathbb{G}'})\}$ where $\rho_{\mathbb{G}'}$ are all the terms (represented as polynomials) provided by the assumption in group $\mathbb{G}' \in \{\mathbb{G}, \mathbb{H}\}$. Since $\mathsf{c_1} = \mathsf{c'_1} = 1$ and assuming that $\mathcal{D}$ maps at least one index into each group, we get $\delta_{\mathbb{G}'} = (d_1 - 1)(d_2 - 1) + 3$. Using the pairing oracle Pair, the adversary can further create polynomials of degree at most $\delta_{\mathbb{G}_T} = \delta_{\mathbb{G}} + \delta_{\mathbb{H}} = 2(d_1 - 1)(d_2 - 1) + 6$.

To bound the difference between the symbolic simulation via polynomials and the actual simulation, we apply the Schwartz-Zippel Lemma. We know that $|\mathcal{L}^*| = |\mathcal{L}_{\mathbb{G}} \cup \mathcal{L}_{\mathbb{H}} \cup \mathcal{L}_{\mathbb{G}_T}| \leq n_{\text{op}} + n_{\text{pair}} + 3 + 3\max\{d_1^2 d_2, d_1 d_2^2\}$, where the last term bounds the number of terms provided by the assumptions. Then an inconsistency occurs with probability at most $\binom{|\mathcal{L}^*|}{2} \cdot \delta_{\mathbb{G}_T}/p$, which yields the bound in the theorem.

Symbolic Security. It remains to show that in the symbolic model the challenge bit is information-theoretically hidden from the adversary's view. This can be observed by looking at the combination of terms from the assumption and that no linear combination allows to recover the polynomial xyz. □

D.2 Proof of Theorem 4

Proof. We simulate the security game using polynomials provided by $\Gamma_{\text{PES-ISA}}$ which describes exactly what is happening in the exponents of master public keys, ciphertexts and secret keys. The main technical difficulty and difference to previous work lies in answering adaptive corruption queries. Luckily, our setting allows to handle corruptions similar to [KPRR23] who analyze Diffie-Hellman assumptions with corruptions. Since in our case, corruptions can also only asked for group elements created by the challenger, we do not need to answer arbitrary discrete logarithm queries, which makes the proof in [BFP21] more involved. So instead of applying the Schwartz-Zippel Lemma once, we will use it after each corruption. We will now describe the simulation in more detail.

At the beginning of the game, the challenger initializes lists $\mathcal{L}_{\mathbb{G}}$ and $\mathcal{L}_{\mathbb{H}}$ with the generators, i.e., $\mathcal{L}_{\mathbb{G}}[1] = \mathcal{L}_{\mathbb{H}}[1] = 1$. List $\mathcal{L}_{\mathbb{G}_T}$ is initialized as empty. The challenger also picks a random bit δ to simulate challenge queries.

The adversary A now gets access to all oracles of the game, as well as GGM oracles Op and Pair as described above. When A queries $\text{MPK}(i)$, then Param_i is run and the monomials are appended to the corresponding lists and the new indices are provided to A. Similarly, when Chall or KeyGen are run, the polynomials are also appended so that they can be used to compute new group elements. If a polynomial repeats, the list is not extended, but A is given the index where this polynomial is stored. During these queries, the challenger may also create new formal variables representing group elements that are generated via hashing in the actual scheme. The adversary can ask for their handles by querying the random oracles. If the random oracles are queried on inputs that have not been used yet, a new variable is appended to the list and its index in $\mathcal{L}_{\mathbb{G}}$ (or $\mathcal{L}_{\mathbb{H}}$) is returned to A.

Simulating corruptions. So far, the simulation happens on a symbolic level only. However, during a corruption, the challenger will have to assign a value to the formal variable that A asks to corrupt. The challenger thus draws an element uniformly from $\mathbb{Z}_p$. This might reveal an inconsistency in the simulation. For example, consider a corruption of a formal variable b, representing a group element in $\mathbb{G}$. Whenever there exist two indices i, i' such that $\mathcal{L}_{\mathbb{G}}[i] \neq \mathcal{L}_{\mathbb{G}}[i']$, but (partly) evaluating the polynomials on $b \in_R \mathbb{Z}_p$ results in $\mathcal{L}_{\mathbb{G}}[i] = \mathcal{L}_{\mathbb{G}}[i']$. In this case, the challenger will abort. Luckily, since b is uniform, we can bound such an inconsistency using Lemma 13. For this we look at the combined list

$\mathcal{L}^* = \mathcal{L}_{\mathbb{G}} \cup \mathcal{L}_{\mathbb{H}} \cup \mathcal{L}_{\mathbb{G}_T}$. Note that due to the structure of $\Gamma_{\text{PES-ISA}}$, polynomials in $\mathcal{L}_{\mathbb{G}}$ and $\mathcal{L}_{\mathbb{H}}$ have at most degree 2. Thus, polynomials in $\mathcal{L}_{\mathbb{G}_T}$ and $\mathcal{L}^*$ have at most degree 4. We let $\ell_1 = |\mathcal{L}^*|$ be the length of $\mathcal{L}^*$ at the time the first corruption query happens. Then the challenger will only abort with probability $\binom{\ell_1}{2} \cdot \frac{4}{p}$. If no such inconsistency has happened, the challenger updates all lists by (partly) evaluating the polynomials on the corrupted values and proceeds with the simulation. Subsequent corruptions are handled in the same way. We define ℓ_t to be the length of $\mathcal{L}^*$ at the time of the t-th corruption. In order to avoid overcounting, we want to make use of the fact that we already know that there are no collisions in the previous ℓ_{t-1} entries. We can thus bound the probability of the challenger aborting on the t-th corrupt query by $(\binom{\ell_t - \ell_{t-1}}{2} + \ell_{t-1} \cdot (\ell_t - \ell_{t-1})) \cdot \frac{4}{p}$. We will look closer at the overall abort probability and the polynomials stored in the final list $\mathcal{L}^*$ in the next paragraph.

Applying the SSSP. Finally, A returns a bit δ'. We will apply Schwartz-Zippel one last time, now choosing random values for all remaining formal variables. Let $\ell := \ell_{n_{\text{cor}}+1}$ be the length of $\mathcal{L}^*$ at the end of the game. Then, the probability of the challenger aborting at any time during the game or after the final assignment is bounded by

$$\sum_{t \in [n_{\text{cor}}]} \left(\binom{\ell_{t+1} - \ell_t}{2} + \ell_t \cdot (\ell_{t+1} - \ell_t)) \right) \cdot \frac{4}{p} \leq \frac{4\ell^2}{p} \ .$$

We still have to argue that the probability that $\delta' = \delta$ is exactly $1/2$. Note that if the adversary violated the predicate in any of its queries, the challenger returns a random bit, so we can now restrict to an adversary that does not violate the predicate. In order to argue that δ is independent of the adversary's view, we will use the SSSP and in particular the fact that it implies strong MK-MC security under corruptions by combining Theorems 1 and 2. Observing that before the final assignment we have $\mathcal{L}^* \subset \text{span}(\mathbf{p_{all}})$, where $\mathbf{p_{all}}$ is as defined in Definition 12, we conclude that $\Pr[\delta' = \delta] = 1/2$.

Further, we can upper bound ℓ by $(2 + n_{\text{mpk}}|\mathsf{MPK}| + n_c|\mathsf{CT}| + n_k|\mathsf{SK}| + n_{\text{ro}} + n_{\text{op}} + n_{\text{pair}})$, where $|\cdot|$ denotes the maximum length of that instance queried by A, which yields the bound stated in the theorem. □

E Existing pair encoding schemes

We review several existing pair encoding schemes from the literature and define explicit molds for those so that we can instantiate them with the ISABELLA compiler. In this section, we instantiate the pair encoding schemes with one overarching $\mathbf{c}'$ that also covers $\mathbf{c}''$ to simplify proving security (see Remark 2).

E.1 Decentralized large-universe ABE based on RW15 and AC17

We give a description of the PES that was given in [Ven23], which is based on the Rouselakis-Waters (RW15) [RW15] scheme and uses the multi-use techniques by Agrawal and Chase [AC17b] to speed up the decryption algorithm. For this scheme, we define the multi-use mapping $\tau\colon [n_1] \to [m]$, which maps the rows associated with the same attributes to different integers, i.e., $m = \max_{j \in [n_1]} |\rho^{-1}(\rho(j))|$, and τ is injective on the sub-domain $\rho^{-1}(\rho(j)) \subseteq [n_1]$. Let $\tilde{\mathcal{S}}$ be the set of authorities in the first entry of the attributes in $\mathcal{S}$.

Definition 21 (Decentralized large-universe CP-ABE from FDH)**.** Let $\mathcal{U}$ denote a universe of attributes and let n_{aut} denote a number of authorities. We define a PES-ISA for the predicate $P_{\text{MA-CP-mu}}\colon \mathcal{X}_{\text{MA}} \times \mathcal{Y}_{\text{MA}} \to \{0,1\}$, where $\mathcal{X}_{\text{MA}} = \{(\mathbf{A}, \rho, \tilde{\rho}, \tau) \mid (\mathbf{A}, \rho) \in \mathcal{X}_{\text{CP-basic}}, \tilde{\rho}\colon [n_1] \to [n_{\text{aut}}], \tau \text{ is a multi-use map}\}$ and $\mathcal{Y}_{\text{MA}} = \{\mathcal{S} \mid \mathcal{S} \subseteq [n_{\text{aut}}] \times \mathcal{U}\}$.

- $\mathrm{Param}(\mathcal{U}) \to (n_\alpha, n_b, \boldsymbol{\alpha}, \mathbf{b})$: Let $\{\mathcal{A}_l\}_{[n_{\mathrm{aut}}]}$ be the authority identifiers, and $n_\alpha = n_{\mathrm{aut}}$ and $n_b = 2n_{\mathrm{aut}} + |\mathcal{U}|$, $\boldsymbol{\alpha} = (\{\alpha_l\}_{l\in[n_{\mathrm{aut}}]}$, and $\mathbf{b} = (\{b_l, b'_l\}_{l\in[n_{\mathrm{aut}}]}, \{b''_{\mathsf{att}}\}_{\mathsf{att}\in\mathcal{U}})$.

- $\mathrm{EncKey}(\mathcal{S}) \to (m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \mathbf{b}))$: We set $m_1 = |\tilde{\mathcal{S}}|+1$, $m_2 = 0$ and $\mathbf{k} = (\{\{k_{1,l} = \alpha_l + r_{\mathsf{GID}} b_l + r_l b'_l, k_{2,(l,\mathsf{att})} = r_l b''_{\mathsf{att}}\}_{(l,\mathsf{att})\in\mathcal{S}})$.

- $\mathrm{EncCt}(\mathbf{A}, \rho, \tilde{\rho}, \tau) \to (w_1, w_2, w'_2, c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}))$: We set $w_1 = m+n_1$, $w_2 = n_2-1$, $w'_2 = n_2-1$, $c_M = \tilde{s}$, $\mathbf{c} = (\{c_{1,j} = \mu_j + s_j b_{\tilde{\rho}(j)}, c_{2,j} = s_j b'_{\tilde{\rho}(j)} + s'_{\tau(j)} b''_{\rho(j)}\}_{j\in[n_1]})$ and $\mathbf{c}' = (\{c'_j = \lambda_j + \alpha_{\tilde{\rho}(j)} s_j\}_{j\in[n_1]})$, where $\lambda_j = A_{j,1}\tilde{s} + \sum_{k\in[2,n_2]} A_{j,k}\hat{v}_k$ and $\mu_j = \sum_{k\in[2,n_2]} A_{j,k}\hat{v}'_k$.

Supporting unbounded authorities. The use of n_{aut} to denote the number of authorities is purely syntactical and for simplicity of notation. Instead of mapping the rows and attributes to the interval $[n_{\mathrm{aut}}]$, we can also map to our authority identifier universe $\mathcal{AID}$. The proofs extend readily to this notational change.

Full-domain hashes and distributions. We set $\mathcal{F}(b''_{\mathsf{att}}) = (1, \mathsf{att})$ for all $\mathsf{att} \in \mathcal{U}$ and $\mathcal{F}(r_{\mathsf{GID}}) = (2, \mathsf{GID})$. Furthermore, any distributions that take correctness of the FDHs into account is suitable. For our concrete instantiations below, we optimize the distribution of the keys and ciphertexts so that encryption and decryption efficiency are optimized (following the ABE Squared approach [dlPVA22]). Almost all ciphertext components can be placed in $\mathbb{G}$, except s'_ℓ for $\ell \in [m]$, which needs to be paired with $k_{2,\mathsf{att}}$ and contains a common variable that is generated via an FDH and must thus be placed in $\mathbb{G}$. For the scheme instantiated with the Ven23 compiler, the $\mathbf{c}'$ encodings need to be placed in $\mathbb{G}_T$, while with the ISABELLA compiler, we can move those encodings to $\mathbf{c}''$ and instantiate in $\mathbb{G}$. (Note that we have to place it in the same group as s_j, which is in $\mathbb{G}$.)

SPM for multi-authority support. We can define a split-predicate mold for multi-authority CP-ABE for the scheme above. In particular, we define a sub-predicate space $\mathcal{Y}_{\mathrm{SA,l}} = \{\mathcal{S}_l \in \mathcal{Y}_{\text{CP-basic}}\}$ for authority $l \in [n_{\mathrm{aut}}]$ of the key predicate space $\mathcal{Y}_{\mathrm{MA}}$. We define the aggregation function as $\mathsf{Agg}_{\mathrm{key},\mathcal{J}'}(\{\mathcal{S}_l\}_{l\in\mathcal{J}'}) = \{(l, \mathsf{att}) \mid l \in \mathcal{J}', \mathsf{att} \in \mathcal{S}_l\}$ (with $\mathcal{J}' \subseteq [n_{\mathrm{aut}}]$). The sub-predicate spaces are connected together with $\mathrm{aux}_y = \mathsf{GID}$. The split algorithms are indexed in $l \in [n_{\mathrm{aut}}]$:

- $\mathrm{SplitParam}_l(\mathcal{U}) \to (\boldsymbol{\alpha}_l = \alpha_l, \boldsymbol{\beta}_l = \emptyset, \mathbf{b}_l = (b_l, b'_l, \{b'_{\mathsf{att}}\}_{\mathsf{att}\in\mathcal{U}}))$;

- $\mathrm{SplitEncKey}_l(\mathcal{S}_l, \mathsf{GID}) \to \mathbf{k}_l = (k_{1,l} = \alpha_l + r_{\mathsf{GID}} b_l + r_l b'_l, \{k_{2,(l,\mathsf{att})} = r_l b''_{\mathsf{att}}\}_{\mathsf{att}\in\mathcal{S}_l})$.

We can also generate $\{b'_{\mathsf{att}}\}_{\mathsf{att}\in\mathcal{U}}$ in a separate instance of $\mathrm{SplitParam}_0$, but note that we would have to alter our multi-authority construction in Section 6 accordingly to support this, and since these variables are generated via an FDH, this distinction is purely syntactical. The effect of corruptions is as explained in Section 6.1.

Instantiation with the Ven23 and ISABELLA compilers. To illustrate the effectiveness of our framework, we also give concrete descriptions of this scheme with the syntax for multi-authority ABE. The instantiations of the PES-ISA in Definition 21 with the above descriptions is in Fig. 2, which describes the scheme generated with the ISABELLA compiler, with differences highlighted when the scheme is instantiated with the Ven23 compiler. We provide further evaluation in Appendix G, comparing the schemes also to RW15 [RW15] and AG21 [AG21].

$\underline{\mathsf{GlobalSetup}(1^\lambda)}$. Pick a pairing group $\mathcal{PG} := (\mathbb{G}, \mathbb{H}, \mathbb{G}_T, p, e, g, h)$ and hash functions $\mathsf{H}_1 : \mathcal{U} \to \mathbb{G}$ and $\mathsf{H}_2 : \mathcal{GID} \to \mathbb{H}$. Define $\mathsf{GP} := (\mathcal{PG}, \mathsf{H}_1, \mathsf{H}_2)$.

$\underline{\mathsf{AuthoritySetup}(\mathsf{GP}, \mathcal{A}_l)}$. Pick $\beta_l, b_l, b'_l \in_R \mathbb{Z}_p$. Compute the master public key of authority $\mathcal{A}_l$ as

$$\mathsf{MPK}_{\mathcal{A}_l} := \left(\mathsf{GP}, A_l = g^{\beta_l}, B_l = g^{b_l}, B'_l = g^{b'_l}\right) .$$

Let $\mathsf{MSK}_{\mathcal{A}_l} := (\beta_l, b_l, b'_l)$ be the master secret key of authority $\mathcal{A}_l$.

$\underline{\mathsf{KeyGen}(\mathcal{A}_l, \mathsf{MSK}_{\mathcal{A}_l}, \mathsf{GID}, \mathcal{S}_{\mathcal{A}_l} \subseteq \mathcal{U}_{\mathcal{A}_l})}$. Pick $r_l \in_R \mathbb{Z}_p$. Compute

$$K_{0,l} := h^{r_l} \qquad K_{1,l} := h^{\beta_l} \cdot \mathsf{H}_2(\mathsf{GID})^{b_l} \cdot h^{r_l b'_l} \qquad K_{2,\mathsf{att}} := \mathsf{H}_1(\mathsf{att})^{r_l}$$

for each $\mathsf{att} \in \mathcal{S}_{\mathcal{A}_l}$. Output $\mathsf{SK}_{\mathcal{A}_l,\mathsf{GID}} := (K_{0,l}, K_{1,l}, \{K_{2,\mathsf{att}}\}_{\mathsf{att}\in\mathcal{S}_{\mathcal{A}_l}})$.

$\underline{\mathsf{Encrypt}(\{\mathcal{A}_l, \mathsf{MPK}_{\mathcal{A}_l}\}_{l\in\tilde{\rho}([n_1])}, (\mathbf{A}, \rho, \tilde{\rho}, \tau))}$. Let $\mathsf{MPK}_{\mathcal{A}_l} = (\mathsf{GP}, A_l, B_l, B'_l)$. Pick $\tilde{s} \in_R \mathbb{Z}_p$, $\mathbf{s} \in_R \mathbb{Z}_p^{n_1}$, $\mathbf{s}' \in_R \mathbb{Z}_p^m$, $\mathbf{v}, \mathbf{v}' \in_R \mathbb{Z}_p^{n_2-1}$. For $\ell \in [m]$ and for each row $j \in [n_1]$ compute

$$C_{1,\ell} := h^{s'_\ell} \qquad C_{2,j} := g^{s_j} \qquad C_{3,j} := g^{\mu_j} \cdot B_{\tilde{\rho}(j)}^{s_j}$$

$$C_{4,j} := (B'_{\tilde{\rho}(j)})^{s_j} \cdot \mathsf{H}_1(\rho(j))^{s'_{\tau(j)}} \qquad C_{5,j} := g^{\lambda_j} \cdot A_{\tilde{\rho}(j)}^{s_j} ,$$

where $\lambda_j = \mathbf{A}_j(\tilde{s}\|\mathbf{v})^\top$ and $\mu_j = \mathbf{A}_j(0\|\mathbf{v}')^\top$. Further, compute $K := e(g,h)^{\tilde{s}}$ and output (K, CT) with $\mathsf{CT} := ((C_{1,\ell})_{\ell\in[m]}, (C_{2,j}, C_{3,j}, C_{4,j}, C_{5,j})_{j\in[n_1]})$.

$\underline{\mathsf{Decrypt}(\{\mathcal{A}_l, \mathsf{MPK}_{\mathcal{A}_l}, \mathsf{SK}_{\mathcal{A}_l,\mathsf{GID}}\}_{l\in\tilde{\rho}_{\mathcal{S}}(\mathcal{S})}, \mathsf{CT})}$. If $\mathcal{S}$ satisfies $(\mathbf{A}, \rho, \tilde{\rho}, \tau)$, there exist constants $\{\varepsilon_j\}_{j\in\Upsilon}$ s.t. $\sum_{j\in\Upsilon} \varepsilon_j \mathbf{A}_j = (1, 0, \ldots, 0)$, where $\Upsilon = \{j \in [n_1] \mid \rho(j) \in \mathcal{S}\}$. Let $K_{\mathcal{A}_l,\mathsf{GID}} = (K_{0,l}, K_{1,l}, K_{2,\mathsf{att}})$. Compute

$$Z := e\left(\prod_{j\in\Upsilon} C_{5,j}^{\varepsilon_j}, h\right) \cdot \prod_{l\in\tilde{\rho}_{\mathcal{S}}(\mathcal{S})} \frac{e\left(\prod_{j\in\Upsilon,\tilde{\rho}(\rho(j))=l} C_{4,j}^{\varepsilon_j}, K_{0,l}\right)}{e\left(\prod_{j\in\Upsilon,\tilde{\rho}(\rho(j))=l} C_{2,j}^{\varepsilon_j}, K_{1,l}\right)} \cdot \frac{e\left(\prod_{j\in\Upsilon} C_{3,j}^{\varepsilon_j}, \mathsf{H}_2(\mathsf{GID})\right)}{\prod_{\ell\in[m]} e\left(\prod_{j\in\Upsilon,\tau(j)=\ell} K_{2,\rho(j)}^{\varepsilon_j}, C_{1,\ell}\right)}$$

and output $M = C_M / Z$.

Figure 2: The decentralized CP-ABE scheme for monotone span programs ($\mathbf{A} \in \mathbb{Z}_p^{n_1\times n_2}$, $\rho\colon [n_1] \to \mathcal{U}, \tilde{\rho} : [n_1] \to [n_{\text{aut}}]$) based on the PES-ISA in Definition 21, instantiated with the ISABELLA compiler. Let $\tilde{\rho}_{\mathcal{S}}\colon \mathcal{S} \to [n_{\text{aut}}]$ be the function that maps each attribute to an authority. We further define $m = \max_{j\in[n_1]} |\rho(j)|$ corresponding to maximum number of times an attribute is used in $\mathbf{A}$, and $\tau\colon [n_1] \to [m]$ maps each row that is associated with the same attribute injectively in $[m]$, i.e., for $j \neq j'$ with $\rho(j) = \rho(j')$, we have $\tau(j) \neq \tau(j')$. We have highlighted the differences between this instantiation and the Ven23 instantiation in red.

E.2 ABE with attribute-wise key generation based on RW13

We give the PES for attribute-wise key generation (based on the CP-ABE scheme by Rouselakis and Waters (RW13) [RW13]) as specified in [Ven23]. Via our framework, this scheme is adaptively secure in GGM. In contrast, this scheme was only statically secure in [Ven23].

Definition 22 (RW13 with attribute-wise key generation)**.** We define the PES-ISA for the predicate $P\colon \mathcal{X}_{\text{CP-basic}} \times \mathcal{Y}_{\text{CP-basic}} \to \{0,1\}$ that will support attribute-wise key generation as follows.

- $\text{Param}(\emptyset) \to (n_\alpha, n_b, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b})$: We set $n_\alpha = 1$, $n_\beta = 0$ and $n_b = 4$, where $\boldsymbol{\alpha} = \alpha$, and $\mathbf{b} = (b, b', b_0, b_1)$.

- $\mathrm{EncKey}(\mathcal{S}) \to (m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{b}))$: We set $m_1 = |\mathcal{S}| + 1$ and $m_2 = 0$, where $\mathbf{k} = (\{k_1 = \alpha + r_{\mathsf{GID}} b, k_{\mathsf{att}} = r_{\mathsf{GID}} b' + r_{\mathsf{att}}(b_0 + x_{\mathsf{att}} b_1)\}_{\mathsf{att} \in \mathcal{S}})$ such that x_{att} is the integer representation of att in $\mathbb{Z}_p$.
- $\mathrm{EncCt}(\mathbf{A}, \rho) \to (w_1, w_2, w_2', c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \mathbf{c}''(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\beta})$: We set $w_1 = n_1$, $w_2 = n_2 - 1$, $w_2' = 0$, $c_M = \alpha s$,

$$\mathbf{c} = \left(\{c_{1,j} = \lambda_j + s_j b', c_{2,j} = s_j(b_0 + \rho(j) b_1)\}_{j \in [n_1]}\right),$$

and $\mathbf{c}' = \mathbf{c}'' = \emptyset$, where $\lambda_j = A_{j,1} s b + \sum_{k \in [2, n_2]} A_{j,k} \hat{v}_k$.

Full-domain hashes and distributions. We set $\mathcal{F}(r_{\mathsf{GID}}) = (1, \mathsf{GID})$. Furthermore, any distributions that take correctness of the FDH into account are suitable.

SPM for attribute-wise key generation. We can define a split-predicate mold for CP-ABE with attribute-wise key generation for the scheme above. In particular, we define a sub-predicate space $\mathcal{Y}_{\mathsf{att}} = \{\mathsf{att}\}$ for each $\mathsf{att} \in \mathcal{U}$ of the key predicate space $\mathcal{Y}_{\text{CP-basic}}$. We define the aggregation function as $\mathsf{Agg}_{\mathrm{key},\mathcal{S}}(\{\mathsf{att}\}_{\mathsf{att} \in \mathcal{S}}) = \mathcal{S}$ (with $\mathcal{S} \subseteq \mathcal{U}$). The sub-predicate spaces are connected together with $\mathrm{aux}_y = \mathsf{GID}$. The split parameter generation has only a single instance and the split key generation algorithm is indexed in 0 and $\mathsf{att} \in \mathcal{U}$:

- $\mathrm{SplitParam}_1(\emptyset) \to (\boldsymbol{\alpha} = \alpha, \boldsymbol{\beta} = \emptyset, \mathbf{b} = (b, b', b_0, b_1))$;
- $\mathrm{SplitEncKey}_0(\emptyset, \mathsf{GID}) \to \mathbf{k}_0 = \alpha + r_{\mathsf{GID}} b$;
- $\mathrm{SplitEncKey}_{\mathsf{att}}(\mathsf{att}, \mathsf{GID}) \to \mathbf{k}_{\mathsf{att}} = r_{\mathsf{GID}} b' + r_{\mathsf{att}}(b_0 + x_{\mathsf{att}} b_1)$.

F New pair encoding schemes

F.1 Decentralized large-universe KP-ABE from FDH

We give a new decentralized KP-ABE scheme, which allows the encrypting user to determine for which authorities the decrypting user needs to have secret keys. Not only does this give the encrypting user more control over the authorities they "trust", this also makes the scheme more resilient to authorities joining and leaving the system. In contrast, existing schemes such as [Cha07, CC09, LHC+11] fix the set of authorities used in the system upon setup.

The predicate $P_{\text{MA-KP-ABE}} \colon \mathcal{X}_{\mathrm{MA}} \times \mathcal{Y}_{\mathrm{MA}} \to \{0, 1\}$ is defined over the ciphertext predicate space $\mathcal{X}_{\mathrm{MA}} = \{(\mathcal{S}, \mathcal{S}_{\mathcal{A}}) \mid \mathcal{S} \subseteq \mathcal{U}, \mathcal{S}_{\mathcal{A}} \subseteq [n_{\mathrm{aut}}]\}$, which denotes pairs consisting of one attribute set and one authority set, and the key predicate space $\mathcal{Y}_{\mathrm{MA}} = \{\{(\mathbf{A}_l, \rho_l)\}_{l \in \mathcal{S}'_{\mathcal{A}}} \mid \mathcal{S}'_{\mathcal{A}} \subseteq [n_{\mathrm{aut}}], (\mathbf{A}_l, \rho_l) \in \mathcal{Y}_{\text{KP-basic}}\}$, which consists of a monotone policy $(\mathbf{A}_l, \rho_l)$ for each authority l in some set of authorities $\mathcal{S}'_{\mathcal{A}}$. The predicate evaluates to true if, for each authority l in the ciphertext authority set $\mathcal{S}_{\mathcal{A}}$, the key policy $(\mathbf{A}_l, \rho_l)$ is satisfied by the ciphertext attribute set $\mathcal{S}$. (By extension, we need it to hold that $\mathcal{S}_{\mathcal{A}} \subseteq \mathcal{S}'_{\mathcal{A}}$.)

Definition 23 (Decentralized large-universe KP-ABE from FDH with flexible authority choice)**.** Let $\mathcal{U}$ denote a universe of attributes. We define a PES-ISA for the predicate $P_{\text{MA-KP-ABE}} \colon \mathcal{X}_{\mathrm{MA}} \times \mathcal{Y}_{\mathrm{MA}} \to \{0, 1\}$, where $\mathcal{X}_{\mathrm{MA}} = \{(\mathcal{S}, \mathcal{S}_{\mathcal{A}}) \mid \mathcal{S} \subseteq \mathcal{U}, \mathcal{S}_{\mathcal{A}} \subseteq [n_{\mathrm{aut}}]\}$ and $\mathcal{Y}_{\mathrm{MA}} = \{\{(\mathbf{A}_l, \rho_l)\}_{l \in \mathcal{S}'_{\mathcal{A}}} \mid \mathcal{S}'_{\mathcal{A}} \subseteq [n_{\mathrm{aut}}], (\mathbf{A}_l, \rho_l) \in \mathcal{Y}_{\text{KP-basic}}\}$. The predicate $P_{\text{MA-KP-ABE}}$ evaluates $P_{\text{MA-KP-ABE}}(x, y) = 1$ with $x = (\mathcal{S}, \mathcal{S}_{\mathcal{A}})$ and $y = \{(\mathbf{A}_l, \rho_l)\}_{l \in \mathcal{S}'_{\mathcal{A}}}$ iff $\mathcal{S}_{\mathcal{A}} \subseteq \mathcal{S}'_{\mathcal{A}}$ and, for all $l \in \mathcal{S}_{\mathcal{A}}$, the set $\mathcal{S}$ satisfies $(\mathbf{A}_l, \rho_l)$.

- $\mathrm{Param}(\mathcal{U}) \to (n_\alpha, n_b, \boldsymbol{\alpha}, \mathbf{b})$: Let $\{\mathcal{A}_l\}_{l \in [n_{\mathrm{aut}}]}$ be the authorities, and $n_\alpha = n_{\mathrm{aut}} + 1$ and $n_b = n_{\mathrm{aut}} + |\mathcal{U}|$, $\boldsymbol{\alpha} = (\{\alpha_l\}_{l \in [n_{\mathrm{aut}}]})$, and $\mathbf{b} = (\{b_l\}_{l \in [n_{\mathrm{aut}}]}, \{b'_{\mathsf{att}}\}_{\mathsf{att} \in \mathcal{U}})$, where $\mathcal{U}$ denotes the universe.

- $\mathrm{EncKey}(\mathbb{A}) \to (m_1, m_2, \mathbf{k}(\mathbf{r}, \hat{\mathbf{r}}, \boldsymbol{\alpha}, \mathbf{b}))$: Parse $\mathbb{A} = \{(\mathbf{A}_l, \rho_l)\}_{l \in \mathcal{S}'_{\mathcal{A}}}$ with $\mathcal{S}'_{\mathcal{A}} \subseteq [n_{\mathrm{aut}}]$ and $\mathbf{A}_l \in \mathbb{Z}_p^{n_{1,l} \times n_{2,l}}$. We set $m_1 = \sum_l n_{1,l}$, $m_2 = \sum_l n_{2,l}$, $\mathbf{k} = (\{k_{j,l} = \lambda_{l,j} + r_{j,l} b'_{\rho_l(j)}\}_{l \in \mathcal{S}'_{\mathcal{A}}, j \in [n_{1,l}]})$, where $\lambda_{l,j} = A_{l,j,1}(\alpha_l + r_{\mathsf{GID}} b_l) + \sum_{k \in [2,n_{2,l}]} A_{l,j,k} \hat{v}_{l,k}$.
- $\mathrm{EncCt}(\mathcal{S}, \mathcal{S}_{\mathcal{A}}) \to (w_1, w_2, w'_2, c_M, \mathbf{c}(\mathbf{s}, \hat{\mathbf{s}}, \mathbf{b}), \mathbf{c}'(\mathbf{s}, \tilde{\mathbf{s}}, \boldsymbol{\alpha}))$: We set $w_1 = 1$, $w_2 = 0$, $w'_2 = 0$, $c_M = \sum_{l \in \mathcal{S}_{\mathcal{A}}} \alpha_l s$, where $\mathcal{S}_{\mathcal{A}} \subseteq [n_{\mathrm{aut}}]$, and $\mathbf{c} = (\{c_{1,l} = s b_l + v_l, c_{\mathsf{att}} = s b'_{\mathsf{att}}\}_{l \in \mathcal{S}_{\mathcal{A}}, \mathsf{att} \in \mathcal{S}})$, such that $v_{l_{\max}} = -\sum_{l \in \mathcal{S}_{\mathcal{A}} \setminus \{l_{\max}\}} v_l$ and $\mathbf{c}' = \emptyset$, where $l_{\max} = \max_{l \in \mathcal{S}_{\mathcal{A}}} l$.

Full-domain hashes and distributions. We set $\mathcal{F}(b'_{\mathsf{att}}) = (1, \mathsf{att})$ for all $\mathsf{att} \in \mathcal{U}$ and $\mathcal{F}(r_{\mathsf{GID}}) = (2, \mathsf{GID})$. Furthermore, any distributions that take correctness of the FDHs into account is suitable. Note that we can support a more efficient instantiation by moving the $\boldsymbol{\alpha}$ variables to $\boldsymbol{\beta}$ and the $\mathbf{c}'$ polynomials to $\mathbf{c}''$.

Split-predicate mold for multi-authority support. We can define a split-predicate mold for multi-authority KP-ABE for the scheme above. In particular, we define a sub-predicate space $\mathcal{Y}_{\mathrm{SA,l}} = \{(\mathbf{A}_l, \rho_l) \in \mathcal{Y}_{\text{KP-basic}}\}$ for authority $l \in [n_{\mathrm{aut}}]$ of the key predicate space $\mathcal{Y}_{\mathrm{MA}}$ such that $\mathsf{Agg}_{\mathrm{key}, \mathcal{S}'_{\mathcal{A}}}((\mathbf{A}_l, \rho_l)_{l \in \mathcal{S}'_{\mathcal{A}}}) = \{(\mathbf{A}_l, \rho_l)\}_{l \in \mathcal{S}'_{\mathcal{A}}}$. The sub-predicate spaces are connected together with $\mathrm{aux}_y = \mathsf{GID}$. The split algorithms are indexed in $l \in [n_{\mathrm{aut}}]$:

- $\mathrm{SplitParam}_l(\mathcal{U}) \to (\boldsymbol{\alpha}_l = \alpha_l, \boldsymbol{\beta}_l = \emptyset, \mathbf{b}_l = (b_l, \{b'_{\mathsf{att}}\}_{\mathsf{att} \in \mathcal{U}}))$;
- $\mathrm{SplitEncKey}_l((\mathbf{A}_l, \rho_l), \mathsf{GID}) \to \mathbf{k}_l = (\{k_{j,l} = \lambda_{l,j} + r_{j,l} b'_{\rho_l(j)}\}_{j \in [n_{1,l}]})$.

Pair/correctness. We show how the Pair algorithm functions by "simulating" decryption. If $\mathbb{A} \models (\mathcal{S}, \mathcal{S}_{\mathcal{A}})$, this algorithm determines, for all $l \in \mathcal{S}_{\mathcal{A}} \subseteq \mathcal{S}'_{\mathcal{A}}$, $\Upsilon_l = \{j \in [n_{1,l}] \mid \rho_l(j) \in \mathcal{S}\}$, and $\{\varepsilon_{l,j} \in \mathbb{Z}_p\}_{j \in \Upsilon_l}$ so that $\sum_{j \in \Upsilon_l} \varepsilon_{l,j} \lambda_{l,j} = \alpha_l + r_{\mathsf{GID}} b_l$ (Definition 1), and computes

$$
\begin{aligned}
&\sum_{\substack{l \in \mathcal{S}_{\mathcal{A}} \\ j \in [n_{1,j}]}} \varepsilon_{l,j} \left(s \cdot k_{j,l} - A_{l,j,1} r_{\mathsf{GID}} \cdot c_{1,l} - r_{j,l} \cdot c_{\rho_l(j)} \right) \\
&\quad = \sum_{\substack{l \in \mathcal{S}_{\mathcal{A}} \\ j \in [n_{1,j}]}} \varepsilon_{l,j} \left(\lambda_{l,j} s + r_{j,l} s b'_{\rho_l(j)} - A_{l,j,1} r_{\mathsf{GID}} s b_l - A_{l,j,1} r_{\mathsf{GID}} v_l - r_{j,l} b'_{\rho_l(j)} \right) \\
&\quad = \sum_{\substack{l \in \mathcal{S}_{\mathcal{A}} \\ j \in [n_{1,j}]}} \varepsilon_{l,j} \left(\lambda_{l,j} s - A_{l,j,1} r_{\mathsf{GID}} s b_l - A_{l,j,1} r_{\mathsf{GID}} v_l \right) \\
&\quad = \sum_{l \in \mathcal{S}_{\mathcal{A}}} \left((\alpha_l + r_{\mathsf{GID}} b_l) s - r_{\mathsf{GID}} s b_l - r_{\mathsf{GID}} v_l \right) \\
&\quad = \sum_{l \in \mathcal{S}_{\mathcal{A}}} \left(\alpha_l s + r_{\mathsf{GID}} v_l \right) = \sum_{l \in \mathcal{S}_{\mathcal{A}}} \alpha_l s
\end{aligned}
$$

Corruptions and P_{cor}. The predicate for our scheme is expanded upon corruption of authorities in the following way. Let $\mathfrak{C} \subseteq [n_{\mathrm{aut}}]$ denote a set of corrupted authorities and consider $x = (\mathcal{S}, \mathcal{S}_{\mathcal{A}})$ and $y = \{(\mathbf{A}_l, \rho_l)\}_{l \in \mathcal{S}'_{\mathcal{A}}}$. Then P_{cor} evaluates true if $\mathcal{S}$ satisfies $(\mathbf{A}_l, \rho_l)$ for all $l \notin \mathfrak{C}$. (For $l \in \mathfrak{C}$, the adversary can generate the required key material to satisfy it.)

Security. We prove that the PES-ISA satisfies SSSP.

Lemma 14. *The PES-ISA in Definition 23 satisfies SSSP.*

Proof. We set $d_1 = 1$ and $d_2 = |\mathcal{S}_\mathcal{A} \setminus \mathfrak{C}|$. Let $\mathfrak{C} \subseteq [n_{\text{aut}}]$ denote some set of corrupted authorities.

- $\text{EncB}(\mathcal{S}, \mathcal{S}_\mathcal{A}) \to (\{\mathbf{a}_l, \mathbf{B}_l\}_{l\in[n_{\text{aut}}]}, \{\mathbf{B}'_{\mathsf{att}}\}_{\mathsf{att}\in\mathcal{U}})$, where $\mathbf{a}_l = 0$ and $\mathbf{B}_l = \mathbf{0}^{d_1\times d_2}$ for all $l \in \mathfrak{C} \cup [n_{\text{aut}}] \setminus \mathcal{S}_\mathcal{A}$. Let $l_{\max}$ denote the highest entry in $\mathcal{S}_\mathcal{A} \setminus \mathfrak{C}$. We set:

$$\{\mathbf{a}_l = 1\}_{l\in\mathcal{S}_\mathcal{A}\setminus\mathfrak{C}}\,, \qquad \left\{\mathbf{B}_l = -\,\mathbf{1}_l^{d_1\times d_2}\right\}_{l\in\mathcal{S}_\mathcal{A}\setminus(\mathfrak{C}\cup\{l_{\max}\})}\,, \quad \left\{\mathbf{B}'_{\mathsf{att}} = \mathbf{0}^{d_1\times d_2}\right\}_{\mathsf{att}\in\mathcal{S}}\,,$$

$$\mathbf{B}_{l_{\max}} = \sum_{l\in\mathcal{S}_\mathcal{A}\setminus(\mathfrak{C}\cup\{l_{\max}\})} \mathbf{1}_l^{d_1\times d_2}\,, \qquad \{\mathbf{B}'_{\mathsf{att}} = -\sum_{l\in[n_{\text{aut}}]} \mathbf{1}_l^{d_1\times d_2}\}_{\mathsf{att}\notin\mathcal{S}}\,.$$

- $\text{EncR}((\mathcal{S}, \mathcal{S}_\mathcal{A}), \mathbb{A}) \to (\mathbf{r}_{\mathsf{GID}}, \{\mathbf{r}_{j,l}\}_{l\in\mathcal{S}'_\mathcal{A}}, \{\hat{\mathbf{v}}_{l,k}\}_{l\in\mathcal{S}'_\mathcal{A}, k\in[2,n_{2,l}]})$. Then we have two cases: $\mathcal{S}_\mathcal{A} \subseteq \mathcal{S}'_\mathcal{A}$ and $\mathcal{S}_\mathcal{A} \not\subseteq \mathcal{S}'_\mathcal{A}$. If $\mathcal{S}_\mathcal{A} \subseteq \mathcal{S}'_\mathcal{A}$, we assume without loss of generality that $l' \in \mathcal{S}_\mathcal{A} \setminus \mathfrak{C}$ is such that $(\mathbf{A}_{l'}, \rho_{l'})$ is not satisfied by $\mathcal{S}$ (which must exist because the key predicate does not satisfy the ciphertext predicate). Assume that l' is not the highest entry $l_{\max}$. (The proof is similar for when $l' = l_{\max}$, because we set w'_l to be 1 for all $l \neq l'$.) Then, let $\{w'_l\}_{l\in\mathcal{S}_\mathcal{A}\setminus\mathfrak{C}}$ be such that $w'_l = 1$ for all $l \notin \{l_{\max}, l'\}$ and $w'_{l'} = |\mathcal{S}_\mathcal{A} \setminus \mathfrak{C}| - 1$.

$$\mathbf{r}_{\mathsf{GID}} = \sum_{l\in\mathcal{S}_\mathcal{A}\setminus(\mathfrak{C}\cup\{l_{\max}\})} w'_l\overline{\mathbf{1}}_l^{d_2}\,,$$

$$\left\{\mathbf{r}_{j,l'} = \sum_{k\in[n_{2,l}]} A_{l',j,k}(\mathbf{w}_{l'})_k\overline{\mathbf{1}}_{l'}^{d_2}\right\}_{j\in[n_{1,l'}]}\,, \qquad \{\hat{\mathbf{v}}_{l',k} = (\mathbf{w}_{l'})_k\}_{k\in[2,n_{2,l'}]}\,,$$

$$\left\{\mathbf{r}_{j,l} = \overline{\mathbf{0}}^{d_2}\right\}_{\substack{l\in\mathcal{S}'_\mathcal{A}\setminus\{l'\}\\ j\in[n_{1,l'}]}}\,, \qquad \{\hat{\mathbf{v}}_{l,k} = 0\}_{\substack{l\in\mathcal{S}'_\mathcal{A}\setminus\{l'\}\\ k\in[2,n_{2,l}]}}\,,$$

and the vector $\mathbf{w}_{l'}$ is such that $(\mathbf{w}_{l'})_1 = 1 - w'_{l'}$ and $\mathbf{A}_{l,j}\mathbf{w}_{l'}^\intercal = 0$ for all $j \in [n_{1,l'}]$ with $\rho_{l'}(j) \in \mathcal{S}$.

If $\mathcal{S}_\mathcal{A} \not\subseteq \mathcal{S}'_\mathcal{A}$, then there must exist some $l' \in \mathcal{S}_\mathcal{A}$ that is not in $\mathcal{S}'_\mathcal{A}$ and $\mathfrak{C}$ (or we again have the case that there is some l' such that $(\mathbf{A}_{l'}, \rho_{l'})$ is not satisfied by $\mathcal{S}$, in which case we can fall back on the first part of the proof). Assume, without loss of generality, that $l' \neq l_{\max}$. (The proof is similar for when $l' = l_{\max}$, because we set w'_l to be 1 for all $l \neq l'$.) Then, let $\{w'_l\}_{l\in\mathcal{S}_\mathcal{A}\setminus\mathfrak{C}}$ be such that $w'_l = 1$ for all $l \notin \{l_{\max}, l'\}$ and $w'_{l'} = |\mathcal{S}_\mathcal{A} \setminus \mathfrak{C}| - 1$.

$$\mathbf{r}_{\mathsf{GID}} = \sum_{l\in\mathcal{S}_\mathcal{A}\setminus(\mathfrak{C}\cup\{l_{\max}\})} w'_l\overline{\mathbf{1}}_l^{d_2}\,, \quad \left\{\mathbf{r}_{j,l} = \overline{\mathbf{0}}^{d_2}\right\}_{\substack{l\in\mathcal{S}'_\mathcal{A}\\ j\in[n_{1,l'}]}}\,, \quad \{\hat{\mathbf{v}}_{l,k} = 0\}_{\substack{l\in\mathcal{S}'_\mathcal{A}\\ k\in[2,n_{2,l}]}}\,.$$

- $\text{EncS}(\mathcal{S}, \mathcal{S}_\mathcal{A}) \to (\mathbf{s}, \{\mathbf{v}_l\}_{l\in\mathcal{S}_\mathcal{A}\setminus\{l_{\max}\}})$, where $\mathbf{s} = 1$, $\mathbf{v}_l = \overline{\mathbf{0}}^{d_2}$ for all $l \in \mathfrak{C}$ and $\mathbf{v}_l = \overline{\mathbf{1}}_l^{d_2}$ for all $l \in \mathcal{S}_\mathcal{A} \setminus (\mathfrak{C} \cup \{l_{\max}\})$.

The proof verifies correctly, i.e., filling in the substitutions in the polynomials yields all-zero entries, and c_M is substituted by a nonzero value, i.e., $\sum_{l\in\mathcal{S}_\mathcal{A}} \alpha_l s \;:\; |\mathcal{S}_\mathcal{A}| \neq 0$. For the keys with $\mathcal{S}_\mathcal{A} \subseteq \mathcal{S}'_\mathcal{A}$, we have, for all $l \in \mathcal{S}'_\mathcal{A} \cap (\mathfrak{C} \cup ([n_{\text{aut}}] \setminus \mathcal{S}_\mathcal{A}))$ that

$$k_{j,l} = \left(A_{l,j,1}(\alpha_l + r_{\mathsf{GID}}b_l) \;+ \sum_{k\in[2,n_{2,l}]} A_{l,j,k}\hat{v}_{l,k}\right) + r_{j,l}b'_{\rho_l(j)} \;:$$

$$\left(A_{l,j,1}(0 + \mathbf{0}^{d_1\times d_2}\mathbf{r}_{\mathsf{GID}}) \;+ \sum_{k\in[2,n_{2,l}]} A_{l,j,k}0\right) + \mathbf{B}'_{\rho_l(j)}\overline{\mathbf{0}}^{d_2} = 0\,,$$

for $l \in \mathcal{S}_{\mathcal{A}'} \setminus (\mathfrak{C} \cup \{l'\})$, we have

$$\begin{aligned} k_{j,l} = &\left(A_{l,j,1}(\alpha_l + r_{\mathsf{GID}} b_l) \ + \sum_{k \in [2,n_{2,l}]} A_{l,j,k} \hat{v}_{l,k} \right) + r_{j,l} b'_{\rho_l(j)} \ : \\ &\left(A_{l,j,1} \left(1 + -\mathbf{1}_l^{d_1 \times d_2} \left(\sum_{l \in \mathcal{S}_{\mathcal{A}} \setminus (\mathfrak{C} \cup \{l_{\max}\})} w'_l \overline{\mathbf{1}}_l^{d_2} \right) \right) \ + \sum_{k \in [2,n_{2,l}]} A_{l,j,k} 0 \right) + \mathbf{b}'_{\rho_l(j)} \overline{\mathbf{0}}^{d_2} \\ = &\ A_{l,j,1}(1 - w'_l) = A_{l,j,1}(1-1) = 0, \end{aligned}$$

and for l', we have

$$\begin{aligned} k_{j,l'} = &\left(A_{l',j,1}(\alpha_{l'} + r_{\mathsf{GID}} b_{l'}) \ + \sum_{k \in [2,n_{2,l}]} A_{l',j,k} \hat{v}_{l',k} \right) + r_{j,l'} b'_{\rho_{l'}(j)} \ : \\ &\left(A_{l',j,1} \left(1 - \mathbf{1}_{l'}^{d_1 \times d_2} \left(\sum_{l \in \mathcal{S}_{\mathcal{A}} \setminus (\mathfrak{C} \cup \{l_{\max}\})} w'_l \overline{\mathbf{1}}_l^{d_2} \right) \right) + \sum_{k \in [2,n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k \right) \\ &+ \mathbf{B}'_{\rho_{l'}(j)} \left(\sum_{k \in [n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k \overline{\mathbf{1}}_{l'}^{d_2} \right) \\ = &\left(A_{l',j,1} \left(1 - w'_{l'}\right) \ + \sum_{k \in [2,n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k \right) + \mathbf{B}'_{\rho_{l'}(j)} \left(\mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal \overline{\mathbf{1}}_{l'}^{d_2} \right) \\ = &\sum_{k \in [n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k + \mathbf{B}'_{\rho_{l'}(j)} \left(\mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal \overline{\mathbf{1}}_{l'}^{d_2} \right) \ . \end{aligned}$$

If $\rho_{l'}(j) \in \mathcal{S}$, then we have that

$$\sum_{k \in [n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k + \mathbf{B}'_{\rho_{l'}(j)} \left(\mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal \overline{\mathbf{1}}_{l'}^{d_2} \right) = 0 + \mathbf{0}^{d_1 \times d_2} \left(\mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal \overline{\mathbf{1}}_{l'}^{d_2} \right) = 0 \ .$$

If $\rho_{l'}(j) \notin \mathcal{S}$, then

$$\begin{aligned} &\sum_{k \in [n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k + \mathbf{B}'_{\rho_{l'}(j)} \left(\mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal \overline{\mathbf{1}}_{l'}^{d_2} \right) \\ &\qquad = \sum_{k \in [n_{2,l}]} A_{l',j,k} (\mathbf{w}_{l'})_k \ - \sum_{l \in [n_{\mathrm{aut}}]} \mathbf{1}_l^{d_1 \times d_2} \left(\mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal \overline{\mathbf{1}}_{l'}^{d_2} \right) \\ &\qquad = \mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal - \mathbf{A}_{l',j} (\mathbf{w}_{l'})^\intercal = 0 \ . \end{aligned}$$

Verification of the keys with $\mathcal{S}_{\mathcal{A}} \not\subseteq \mathcal{S}'_{\mathcal{A}}$ such that there exists $l' \in \mathcal{S}_{\mathcal{A}}$ that is not in $\mathcal{S}'_{\mathcal{A}}$ and $\mathfrak{C}$ is similar.

For the ciphertext polynomials with $l \in \mathcal{S}_{\mathcal{A}} \setminus \mathfrak{C}$, we have

$$c_{1,l} = s b_l + v_l \ : \ 1 \cdot (-\mathbf{1}_l^{d_1 \times d_2}) + \overline{\mathbf{1}}_l^{d_2} = -\overline{\mathbf{1}}_l^{d_2} + \overline{\mathbf{1}}_l^{d_2} = \overline{\mathbf{0}}^{d_2}.$$

For $l \in \mathcal{S}_{\mathcal{A}} \cap \mathfrak{C}$, we have

$$c_{1,l} = s b_l + v_l \ : \ 1 \cdot \mathbf{0}^{d_1 \times d_2} + \overline{\mathbf{0}}^{d_2} = \overline{\mathbf{0}}^{d_2}.$$

For the other ciphertext polynomials, we have

$$c_{\mathsf{att}} = sb'_{\mathsf{att}} \ : \ 1 \cdot \mathbf{0}^{d_1 \times d_2} = \overline{\mathbf{0}}^{d_2}.$$

□

F.2 Completely unbounded decentralized CP-ABE with negations

We give a decentralized large-universe CP-ABE scheme that supports negations in a completely unbounded way. This is in contrast to existing such schemes that support negations [AG23, Ven23], which are bounded in the number of key attributes that can be issued per authority [AG23] and the number of re-uses of the same label in the keys [Ven23]. To achieve this, we generalize the decentralized variant of the TKN20 [TKN20] scheme in [Ven23, §5.1] to the unbounded label-use setting (in both the key sets and ciphertext policies). We do this by leveraging the techniques in [Att19], which binds each attribute within a label together with an AND-statement. In this way, the comparison with the negated attribute (within the label) needs to be done for each attribute with that label. Our SSSP proofs below can be seen as a generalization of the proof for the bounded version to the unbounded setting. (Although we note that our proof uses considerably much more "layering" of matrices than the bounded scheme in [Ven23, §5.1].)

The predicate for our scheme is the multi-authority variant of the single-authority predicate denoted as $P_{\text{CP-not}} \colon \mathcal{X}_{\text{CP-not}} \times \mathcal{Y}_{\text{CP-not}} \to \{0,1\}$. That is, compared to the basic CP-ABE predicate, the policies are extended with a mapping that represents negations ρ' and mappings for the labeling ρ_{lab}, and the sets also come with a similar extension to support labeling. In the scheme, we represent the elements in the set as tuples, i.e., $\mathcal{S} \subseteq [n_{\text{aut}}] \times \mathcal{L} \times \mathcal{U} = \{(l, \mathsf{lab}, \mathsf{att}) \mid \mathcal{A}_l \in \mathcal{AID}, \mathsf{lab} \in \mathcal{L}, \mathsf{att} \in \mathcal{U}\}$, and we denote the subset managed by authority l as $\mathcal{S}_l = \{(\mathsf{lab}, \mathsf{att}) \mid (l, \mathsf{lab}, \mathsf{att}) \in \mathcal{S}\}$. We do this to simplify notation. To evaluate the predicate $P_{\text{MA-CP-not}}(x, y)$ to true, it must hold that, for $x = (\mathbf{A}, \rho, \rho', \rho_{\mathsf{lab}}, \tilde{\rho})$ and $y = \mathcal{S}$, and for $\Upsilon = \{j \in \Psi \mid (\rho_{\mathsf{lab}}(j), \rho(j)) \in \mathcal{S}_{\tilde{\rho}(j)}\}$ and $\overline{\Upsilon} = \{j \in \overline{\Psi} \mid (\rho_{\mathsf{lab}}(j), \rho(j)) \notin \mathcal{S}_{\tilde{\rho}(j)} \wedge \exists (\rho_{\mathsf{lab}}(j), \mathsf{att}) \in \mathcal{S}_{\tilde{\rho}(j)}\}$, there exist $\{\varepsilon_j \in \mathbb{Z}_p\}_{j \in \Upsilon \cup \overline{\Upsilon}}$ so that $\sum_{j \in \Upsilon \cup \overline{\Upsilon}} \varepsilon_j \lambda_j = \tilde{s}$.

Definition 24 (Completely unbounded decentralized large-universe CP-ABE with negations)**.** Let $\mathcal{L}$ denote a universe of labels and let n_{aut} denote a number of authorities. We define a PES-ISA for the predicate $P_{\text{MA-CP-not}}$ as follows.

- Param($\mathcal{L}$): Let $\{\mathcal{A}_l\}_{[n_{\text{aut}}]}$ be the authorities. On input the label universe $\mathcal{L}$, we set $n_\alpha = n_{\text{aut}}$ and $n_b = (2 + 4|\mathcal{L}|)n_{\text{aut}}$, where $\boldsymbol{\alpha} = \{\alpha_l\}_{l \in [n_{\text{aut}}]}$, and $\mathbf{b} = (\{b_l, \bar{b}_l, \{b_{l,\mathsf{lab},0}, b_{l,\mathsf{lab},1}, \bar{b}_{l,\mathsf{lab},0}, \bar{b}_{l,\mathsf{lab},1}\}_{\mathsf{lab} \in \mathcal{L}}\}_{l \in [n_{\text{aut}}]})$.
- EncKey($\mathcal{S}, \mathsf{GID}$): We set $m_1 = 2|\mathcal{S}| + 1$, $m_2 = 0$, and

$$\mathbf{k} = \Big(\big\{ k_{1,l,(\mathsf{lab},\mathsf{att})} = \alpha_l + r_{\mathsf{GID}} b_l + r_{l,\mathsf{att}}(b_{l,\mathsf{lab},0} + x_{\mathsf{att}} b_{l,\mathsf{lab},1}),$$
$$k_{2,l,\mathsf{lab}} = \alpha_l + r_{\mathsf{GID}} \bar{b}_l + \bar{r}_{l,\mathsf{lab}} \bar{b}_{l,\mathsf{lab},1},$$
$$k_{3,l,(\mathsf{lab},\mathsf{att})} = \bar{r}_{l,\mathsf{att}}(\bar{b}_{l,\mathsf{lab},0} + x_{\mathsf{att}} \bar{b}_{l,\mathsf{lab},1}) \big\}_{(l,\mathsf{lab},\mathsf{att}) \in \mathcal{S}} \Big)$$

  where $\mathcal{S}_l = \{(\mathsf{lab}, \mathsf{att}) \mid (l, \mathsf{lab}, \mathsf{att}) \in \mathcal{S}\}$, and $\bar{r}_{l,\mathsf{lab}} = \sum_{(\mathsf{lab}',\mathsf{att}) \in \mathcal{S}_l : \mathsf{lab}' = \mathsf{lab}} \bar{r}_{l,\mathsf{att}}$ and x_{att} is the representation of att in $\mathbb{Z}_p$.
- EncCt($\mathbf{A}, \rho, \tilde{\rho}, \rho', \rho_{\mathsf{lab}}$): We set $w_1 = n_1$, $w_2 = n_2 - 1$, $w'_2 = n_2 - 1$, $c_M = \tilde{s}$,

$$\mathbf{c} = \Big(\big\{ c_{1,j} = \mu_j + s_j b_{\tilde{\rho}(j)}, \ \ c_{2,j} = s_j (b_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),0} + x_{\rho(j)} b_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),1}) \big\}_{j \in \Psi},$$

$$\left\{c_{1,j} = \mu_j + s_j \bar{b}_{\tilde{\rho}(j)},\ \ c_{2,j} = s_j(\bar{b}_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),0} + x_{\rho(j)}\bar{b}_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),1})\right\}_{j\in\overline{\Psi}}\Big)$$

and $\mathbf{c}' = (\{c'_j = \lambda_j + \alpha_{\tilde{\rho}(j)} s_j\}_{j\in[n_1]})$, where $\lambda_j = A_{j,1}\tilde{s} + \sum_{k\in[2,n_2]} A_{j,k}\hat{v}_k$ and $\mu_j = \sum_{k\in[2,n_2]} A_{j,k}\hat{v}'_k$, and $\Psi = \{j \in [n_1] \mid \rho'(j) = 1\}$ and $\overline{\Psi} = [n_1] \setminus \Psi$ (i.e., the set of rows associated with the non-negated and negated attributes, respectively), and $\mathbf{s} = (\{s_j\}_{[n_1]})$.

Supporting unbounded authorities and sharing attribute universe. The use of n_{aut} to denote the number of authorities is purely syntactical and for simplicity of notation. Instead of mapping the rows and attributes to the interval $[n_{\text{aut}}]$, we can also map to our authority identifier universe $\mathcal{AID}$. The proofs extend readily to this notational change.

Pair/correctness. We show how the Pair algorithm functions by "simulating" decryption. Let $\mathcal{S}_l = \{(\mathsf{lab}, \mathsf{att}) \mid (l, \mathsf{lab}, \mathsf{att}) \in \mathcal{S}\}$. If $(\mathbf{A}, \rho, \tilde{\rho}, \rho', \rho_{\mathsf{lab}}) \models \mathcal{S}$, this algorithm determines $\Upsilon = \{j \in \Psi \mid (\rho_{\mathsf{lab}}(j), \rho(j)) \in \mathcal{S}_{\tilde{\rho}(j)}\}$, $\overline{\Upsilon} = \{j \in \overline{\Psi} \mid (\rho_{\mathsf{lab}}(j), \rho(j)) \notin \mathcal{S}_{\tilde{\rho}(j)} \wedge \exists(\rho_{\mathsf{lab}}(j), \mathsf{att}) \in \mathcal{S}_{\tilde{\rho}(j)}\}$ and $\{\varepsilon_j \in \mathbb{Z}_p\}_{j\in\Upsilon\cup\overline{\Upsilon}}$ so that $\sum_{j\in\Upsilon\cup\overline{\Upsilon}} \varepsilon_j\lambda_j = \tilde{s}$ (Definition 1), and computes

$$\begin{aligned}
&\sum_{j\in\Upsilon\cup\overline{\Upsilon}} \varepsilon_j c'_j - \sum_{j\in\Upsilon} \varepsilon_j s_j k_{1,\tilde{\rho}(j),(\rho_{\mathsf{lab}}(j),\rho(j))} \\
&\qquad + \sum_{j\in\Upsilon} \varepsilon_j \left(r_{\mathsf{GID}} c_{1,j} + r_{\tilde{\rho}(j),\rho(j)} c_{2,j}\right) - \sum_{j\in\overline{\Upsilon}} \varepsilon_j \left(s_j k_{2,\tilde{\rho}(j),\rho_{\mathsf{lab}}(j)} - r_{\mathsf{GID}} c_{1,j}\right) \\
&\qquad + \sum_{j\in\overline{\Upsilon}} \varepsilon_j \sum_{(\rho_{\mathsf{lab}}(j),\mathsf{att})\in\mathcal{S}_{\tilde{\rho}(j)}} \left(\frac{\bar{r}_{\tilde{\rho}(j),\mathsf{att}} c_{2,j} - s_j k_{3,\tilde{\rho}(j),(\rho_{\mathsf{lab}}(j),\mathsf{att})}}{x_{\mathsf{att}} - x_{\rho(j)}}\right).
\end{aligned}$$

Full-domain hashes and distributions. We set $\mathcal{F}(b_{l,\mathsf{lab},i}) = (4l + i, (\mathcal{A}_l, \mathsf{lab}, 1, i))$ and $\mathcal{F}(\bar{b}_{l,\mathsf{lab},i}) = (4l + i + 2, (\mathcal{A}_l, \mathsf{lab}, 0, i))$ for all $l \in [n_{\text{aut}}], i \in \{0, 1\}, \mathsf{lab} \in \mathcal{L}$. We further set $\mathcal{F}(r_{\mathsf{GID}}) = (2, \mathsf{GID})$. Note that we can support a more efficient instantiation by moving the $\boldsymbol{\alpha}$ variables to $\boldsymbol{\beta}$ and the $\mathbf{c}'$ polynomials to $\mathbf{c}''$.

SPM for multi-authority support. We can define a split-predicate mold for multi-authority CP-ABE for the scheme above. In particular, we define a sub-predicate space $\mathcal{Y}_{\text{SA,l}} = \{\mathcal{S}_l \in \mathcal{Y}_{\text{CP-basic}}\}$ for authority $l \in [n_{\text{aut}}]$ of the key predicate space $\mathcal{Y}_{\text{MA}}$. We define the aggregation function as $\mathsf{Agg}_{\text{key},\mathcal{J}'}(\{\mathcal{S}_l\}_{l\in\mathcal{J}'}) = \{(l, \mathsf{att}) \mid l \in \mathcal{J}', \mathsf{att} \in \mathcal{S}_l\}$ (with $\mathcal{J}' \subseteq [n_{\text{aut}}]$). The sub-predicate spaces are connected together with $\text{aux}_y = \mathsf{GID}$. The split algorithms are indexed in $l \in [n_{\text{aut}}]$:

- $\text{SplitParam}_l(\mathcal{U}) \to \big(\boldsymbol{\alpha}_l = \alpha_l, \boldsymbol{\beta}_l = \emptyset, \mathbf{b}_l = (\{b_{l,\mathsf{lab},0}, b_{l,\mathsf{lab},1}, \bar{b}_{l,\mathsf{lab},0}, \bar{b}_{l,\mathsf{lab},1}\}_{\mathsf{lab}\in\mathcal{L}})\big)$;
- $\text{SplitEncKey}_l(\mathcal{S}_l, \mathsf{GID}) \to \mathbf{k}_l = \big(\{k_{2,l,\mathsf{lab}} = \alpha_l + r_{\mathsf{GID}}\bar{b}_l + \bar{r}_{l,\mathsf{lab}}\bar{b}_{l,\mathsf{lab},1},$
$k_{1,l,(\mathsf{lab},\mathsf{att})} = \alpha_l + r_{\mathsf{GID}} b_l + r_{l,\mathsf{att}}(b_{l,\mathsf{lab},0} + x_{\mathsf{att}} b_{l,\mathsf{lab},1}),$
$k_{3,l,(\mathsf{lab},\mathsf{att})} = \bar{r}_{l,\mathsf{att}}(\bar{b}_{l,\mathsf{lab},0} + x_{\mathsf{att}}\bar{b}_{l,\mathsf{lab},1})\}_{(\mathsf{lab},\mathsf{att})\in\mathcal{S}_l}\big)$.

SPM for MA-ABE and label-wise key generation. We can define a split-predicate mold for multi-authority CP-ABE with label-wise key generation for the scheme above. In particular, we define a sub-predicate space $\mathcal{Y}_{\text{SA,l,lab}} = \mathcal{U}$ for authority $l \in [n_{\text{aut}}]$ and label $\mathsf{lab} \in \mathcal{L}$ of the key predicate space $\mathcal{Y}_{\text{MA}}$. We define the aggregation function as $\mathsf{Agg}_{\text{key},\mathcal{J}'}(\{\mathcal{S}_{l,\mathsf{lab}}\}_{(l,\mathsf{lab})\in\mathcal{J}'}) = \{(l, \mathsf{lab}, \mathsf{att}) \mid (l, \mathsf{lab}) \in \mathcal{J}', \mathsf{att} \in \mathcal{S}_{l,\mathsf{lab}}\}$ (with $\mathcal{J}' \subseteq [n_{\text{aut}}] \times \mathcal{L}$). The sub-predicate spaces are connected together with $\text{aux}_y = \mathsf{GID}$. The split algorithms are indexed in $(l, \mathsf{lab}) \in [n_{\text{aut}}] \times \mathcal{L}$:

- $\mathrm{SplitParam}_l(\mathcal{U}) \to (\boldsymbol{\alpha}_l = \alpha_l, \boldsymbol{\beta}_l = \emptyset, \mathbf{b}_l = (\{b_{l,\mathsf{lab},0}, b_{l,\mathsf{lab},1}, \bar{b}_{l,\mathsf{lab},0}, \bar{b}_{l,\mathsf{lab},1}\}_{\mathsf{lab}\in\mathcal{L}}))$;
- $\mathrm{SplitEncKey}_{(l,\mathsf{lab})}(\mathcal{S}_{l,\mathsf{lab}}, \mathsf{GID}) \to \mathbf{k}_{l,\mathsf{lab}} = (\{k_{2,l,\mathsf{lab}} = \alpha_l + r_{\mathsf{GID}}\bar{b}_l + \bar{r}_{l,\mathsf{lab}}\bar{b}_{l,\mathsf{lab},1},$
  $k_{1,l,(\mathsf{lab},\mathsf{att})} = \alpha_l + r_{\mathsf{GID}}b_l + r_{l,\mathsf{att}}(b_{l,\mathsf{lab},0} + x_{\mathsf{att}}b_{l,\mathsf{lab},1}),$
  $k_{3,l,(\mathsf{lab},\mathsf{att})} = \bar{r}_{l,\mathsf{att}}(\bar{b}_{l,\mathsf{lab},0} + x_{\mathsf{att}}\bar{b}_{l,\mathsf{lab},1})\}_{\mathsf{att}\in\mathcal{S}_{l,\mathsf{lab}}})$.

Security. We prove that the PES-ISA satisfies SSSP.

Lemma 15. *The PES-ISA in Definition 24 satisfies SSSP.*

Proof. We set $d_1 = n_1$ and $d_2 = ((n_1+2)|\rho_{\mathsf{lab}}([n_1])|+1)n_2+1$. Let $\mathfrak{C} \subseteq [n_{\mathrm{aut}}]$ denote some set of corrupted authorities. We also use a special notation from e.g., [VA22, Ven23, VA23] for the column indices that makes the relationship between the input indices and the columns of the substitution matrices clearer. In particular, we map the tuples $(1,k)$, $(2,j,k,\mathsf{lab})$, $(3,k,\mathsf{lab})$, $(4,1)$, $(5,k,\mathsf{lab})$ injectively in the interval $[d_2]$, where $j \in [n_1]$ are the rows of the policy matrix, $k \in [n_2]$ are the columns of the policy matrix, and $\mathsf{lab} \in \rho_{\mathsf{lab}}([n_1])$ are the labels that occur in the ciphertext policy.

- $\mathrm{EncB}(\mathcal{S}) \to (\{\mathbf{a}_l, \mathbf{B}_l, \bar{\mathbf{B}}_l\}_{l\in[n_{\mathrm{aut}}]}, \{\mathbf{B}_{l,\mathsf{lab},i}, \bar{\mathbf{B}}_{l,\mathsf{lab},i}\}_{\mathsf{lab}\in\mathcal{L},i\in\{0,1\}}\}_{l\in[n_{\mathrm{aut}}]})$, where $\mathbf{a}_l = \mathbf{0}^{d_1}$ and $\mathbf{B}_l, \bar{\mathbf{B}}_l, \mathbf{B}_{l,\mathsf{lab},i}, \bar{\mathbf{B}}_{l,\mathsf{lab},i} = \mathbf{0}^{d_1\times d_2}$ for all $l \in \mathfrak{C}$ and $i \in \{0,1\}$, and let $\mathbf{v} \in \mathbb{Z}_p^{n_2}$ (with $v_1 = 1$) be the vector orthogonal to each row $j \in \tilde{\rho}^{-1}(\mathfrak{C})$ associated with a corrupted authority. For all $l \in [n_{\mathrm{aut}}] \setminus \mathfrak{C}$, we set:

$$\mathbf{a}_l = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\\ k\in[n_2]}} A_{j,k}v_k\mathbf{1}_j^{d_1},$$

$$\mathbf{B}_l = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\\ k\in[2,n_2]}} A_{j,k}\left(\mathbf{1}_{j,(1,k)}^{d_1\times d_2} + v_k\mathbf{1}_{j,(1,1)}^{d_1\times d_2}\right) - \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\\ k\in[n_2]}} A_{j,k}v_k\mathbf{1}_{j,(4,1)}^{d_1\times d_2},$$

$$\bar{\mathbf{B}}_l = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\\ k\in[2,n_2]}} A_{j,k}\left(\mathbf{1}_{j,(1,k)}^{d_1\times d_2} + v_k\mathbf{1}_{j,(1,1)}^{d_1\times d_2}\right) - \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\\ k\in[n_2]}} A_{j,k}v_k\mathbf{1}_{j,(4,1)}^{d_1\times d_2},$$

$$\mathbf{B}_{l,\mathsf{lab},0} = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\cap\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} -x_{\rho(j)}A_{j,k}\mathbf{1}_{j,(2,j,k,\mathsf{lab})}^{d_1\times d_2} + \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\setminus\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} A_{j,k}\mathbf{1}_{j,(3,k,\mathsf{lab})}^{d_1\times d_2},$$

$$\bar{\mathbf{B}}_{l,\mathsf{lab},0} = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\cap\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} -x_{\rho(j)}A_{j,k}\mathbf{1}_{j,(2,j,k,\mathsf{lab})}^{d_1\times d_2} + \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\setminus\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} A_{j,k}\mathbf{1}_{j,(5,k,\mathsf{lab})}^{d_1\times d_2},$$

$$\mathbf{B}_{l,\mathsf{lab},1} = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\cap\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} A_{j,k}\mathbf{1}_{j,(2,j,k,\mathsf{lab})}^{d_1\times d_2},$$

$$\bar{\mathbf{B}}_{l,\mathsf{lab},1} = \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\cap\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} A_{j,k}\mathbf{1}_{j,(2,j,k,\mathsf{lab})}^{d_1\times d_2} + \sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\setminus\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}} A_{j,k}\mathbf{1}_{j,(3,k,\mathsf{lab})}^{d_1\times d_2},$$

  where $\Psi_0 = \{j \in [n_1] \mid \rho'(j) = 0\}$ and $\Psi_1 = \{j \in [n_1] \mid \rho'(j) = 1\}$.

- $\mathrm{EncR}((\mathbf{A}, \rho, \tilde{\rho}, \rho', \rho_{\mathsf{lab}}), \mathcal{S})$
  $\to (\mathbf{r}_{\mathsf{GID}}, \{\mathbf{r}_{l,\mathsf{att}}, \bar{\mathbf{r}}_{l,\mathsf{att}}\}_{(l,\mathsf{lab},\mathsf{att})\in\mathcal{S}})$, where

$$\mathbf{r}_{\mathsf{GID}} = \overline{\mathbf{1}}_{(4,1)}^{d_2} - \sum_{k\in[n_2]} w_k\overline{\mathbf{1}}_{(1,k)}^{d_2},$$

$$\mathbf{r}_{l,\mathsf{att}} = \sum_{\mathsf{lab}' \in \chi_{l,\mathsf{att}}} \left(\sum_{\substack{j \in \overline{\Upsilon}_{l,1} \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}') \\ k \in [n_2]}} \frac{w_k \overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')}}{x_{\mathsf{att}} - x_{\rho(j)}} + \sum_{k \in [n_2]} w_k \overline{\mathbf{1}}^{d_2}_{(3,k,\mathsf{lab}')} \right),$$

$$\bar{\mathbf{r}}_{l,\mathsf{att}} = \sum_{\substack{\mathsf{lab}' \in \chi_{l,\mathsf{att}} \\ k \in [n_2]}} w_k \left(\overline{\mathbf{1}}^{d_2}_{(3,k,\mathsf{lab}')} - x_{\mathsf{att}} \overline{\mathbf{1}}^{d_2}_{(5,k,\mathsf{lab}')} + \sum_{j \in \overline{\Upsilon}_{l,0} \cap \rho^{-1}(\mathsf{att})} \overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')} \right),$$

and $\chi_{l,\mathsf{att}} = \{\mathsf{lab} \mid (\mathsf{lab}, \mathsf{att}) \in \mathcal{S}_l\}$, $\overline{\Upsilon}_{l,1} = \{j \in [n_1] \mid \tilde{\rho}(j) = l \wedge \rho'(j) = 1 \wedge (\rho_{\mathsf{lab}}(j), \rho(j)) \notin \mathcal{S}_l\}$, and $\overline{\Upsilon}_{l,0} = \{j \in [n_1] \mid \tilde{\rho}(j) = l \wedge \rho'(j) = 0 \wedge (\rho_{\mathsf{lab}}(j), \rho(j)) \in \mathcal{S}_l\}$, and the vector $\mathbf{w}$ is such that $(\mathbf{w})_1 = -1$ and $\mathbf{A}_j \mathbf{w}^\intercal = 0$ for all j with $\rho'(j) = 1$ and $(\rho_{\mathsf{lab}}(j), \rho(j)) \in \mathcal{S}_{\tilde{\rho}(j)}$ or $\rho'(j) = 0$ and $(\rho_{\mathsf{lab}}(j), \rho(j)) \notin \mathcal{S}_{\tilde{\rho}(j)}$.

- $\mathrm{EncS}(\mathbf{A}, \rho, \tilde{\rho}, \rho', \rho_{\mathsf{lab}}) \to (\{\mathbf{s}_j\}_{j \in [n_1]}, \{\hat{\mathbf{v}}_k, \hat{\mathbf{v}}'_k\}_{k \in [2,n_2]}, \tilde{\mathbf{s}})$, where

$$\tilde{\mathbf{s}} = 1, \qquad \mathbf{s}_j = -\mathbf{1}^{d_1}_j, \qquad \hat{\mathbf{v}}_k = v_k, \qquad \hat{\mathbf{v}}'_k = \overline{\mathbf{1}}^{d_2}_{(1,k)} + v_k \overline{\mathbf{1}}^{d_2}_{(1,1)}.$$

Note that $c_M = \tilde{s}$ is substituted by a nonzero entry. Furthermore, the proof verifies correctly, i.e., substituting the variables for the matrices and vectors above yields all-zero vectors. For the verifications, we define $\Upsilon_{l,0} = \{j \in \Psi_0 \mid \tilde{\rho}(j) = l \wedge (\rho_{\mathsf{lab}}(j), \rho(j)) \notin \mathcal{S}_l\}$ and $\Upsilon_{l,1} = \{j \in \Psi_1 \mid \tilde{\rho}(j) = l \wedge (\rho_{\mathsf{lab}}(j), \rho(j)) \in \mathcal{S}_l\}$. For the key polynomials $k_{1,l,(\mathsf{lab},\mathsf{att})} = \alpha_l + r_{\mathsf{GID}} b_l + r_{l,\mathsf{att}}(b_{l,\mathsf{lab},0} + x_{\mathsf{att}} b_{l,\mathsf{lab},1})$, we first compute the substitutions for $\alpha_l + r_{\mathsf{GID}} b_l$:

$$\begin{aligned}
&\sum_{\substack{j \in \tilde{\rho}^{-1}(l) \\ k \in [n_2]}} A_{j,k} v_k \mathbf{1}^{d_1}_j + \left(\sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \\ k \in [2,n_2]}} A_{j,k} \left(\mathbf{1}^{d_1 \times d_2}_{j,(1,k)} + v_k \mathbf{1}^{d_1 \times d_2}_{j,(1,1)} \right) \right. \\
&\qquad \left. - \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_0 \\ k \in [n_2]}} A_{j,k} v_k \mathbf{1}^{d_1 \times d_2}_{j,(4,1)} \right) \left(\overline{\mathbf{1}}^{d_2}_{(4,1)} - \sum_{k \in [n_2]} w_k \overline{\mathbf{1}}^{d_2}_{(1,k)} \right) \\
&= \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \\ k \in [n_2]}} A_{j,k} v_k \mathbf{1}^{d_1}_j - \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \\ k \in [2,n_2]}} A_{j,k} w_k \mathbf{1}^{d_1}_j + \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \\ k \in [2,n_2]}} A_{j,k} v_k w_1 \mathbf{1}^{d_1}_j - \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_0 \\ k \in [n_2]}} A_{j,k} v_k \mathbf{1}^{d_1}_j \\
&= - \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}^{d_1}_j \\
&= \underbrace{- \sum_{\substack{j \in \Upsilon_{l,1} \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}^{d_1}_j}_{=0} - \sum_{\substack{j \in \overline{\Upsilon}_{l,1} \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}^{d_1}_j .
\end{aligned}$$

This is canceled by $r_{l,\mathsf{att}}(b_{l,\mathsf{lab},0} + x_{\mathsf{att}} b_{l,\mathsf{lab},1})$:

$$\left(\sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} \left(x_{\mathsf{att}} - x_{\rho(j)} \right) A_{j,k} \mathbf{1}^{d_1 \times d_2}_{j,(2,j,k,\mathsf{lab})} + \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \setminus \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} \mathbf{1}^{d_1 \times d_2}_{j,(3,\mathsf{lab})} \right)$$

$$\left(\sum_{\mathsf{lab}' \in \chi_{l,\mathsf{att}}} \left(\sum_{\substack{j \in \overline{\Upsilon}_{l,1} \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}') \\ k \in [n_2]}} \frac{w_k \overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')}}{x_{\mathsf{att}} - x_{\rho(j)}} + \sum_{k \in [n_2]} w_k \overline{\mathbf{1}}^{d_2}_{(3,\mathsf{lab}')} \right) \right)$$

$$= \sum_{\substack{j \in \overline{\Upsilon}_{l,1} \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} \frac{x_{\mathsf{att}} - x_{\rho(j)}}{x_{\mathsf{att}} - x_{\rho(j)}} A_{j,k} w_k \mathbf{1}_j^{d_1} + \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_1 \setminus \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1}$$

$$= \sum_{\substack{j \in \overline{\Upsilon}_{l,1} \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} \frac{x_{\mathsf{att}} - x_{\rho(j)}}{x_{\mathsf{att}} - x_{\rho(j)}} A_{j,k} w_k \mathbf{1}_j^{d_1} + \sum_{\substack{j \in \overline{\Upsilon}_{l,1} \setminus \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1}$$

$$= \sum_{\substack{j \in \overline{\Upsilon}_{l,1} \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1} \ .$$

For the key polynomial $\alpha_l + r_{\mathsf{GID}} \bar{b}_l + \bar{r}_{l,\mathsf{lab}} \bar{b}_{l,\mathsf{lab},1}$, we compute the first part as before, i.e.,

$$\alpha_l + r_{\mathsf{GID}} \bar{b}_l \ : \ - \sum_{\substack{j \in \overline{\Upsilon}_{l,0} \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1} \ ,$$

and the second part as

$$\bar{r}_{l,\mathsf{lab}} \bar{b}_{l,\mathsf{lab},1} \ : \ \left(\sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_0 \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} \mathbf{1}^{d_1 \times d_2}_{j,(2,j,k,\mathsf{lab})} + \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_0 \setminus \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} \mathbf{1}^{d_1 \times d_2}_{j,(3,k,\mathsf{lab})} \right)$$
$$\left(\sum_{(\mathsf{lab}'',\mathsf{att}) \in \mathcal{S}_l : \mathsf{lab}'' = \mathsf{lab}} \left(\sum_{\substack{\mathsf{lab}' \in \chi_{l,\mathsf{att}} \\ k \in [n_2]}} w_k \left(\overline{\mathbf{1}}^{d_2}_{(3,k,\mathsf{lab}')} - x_{\mathsf{att}} \overline{\mathbf{1}}^{d_2}_{(5,k,\mathsf{lab}')} + \sum_{j \in \overline{\Upsilon}_{l,0} \cap \rho^{-1}(\mathsf{att})} \overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')} \right) \right) \right)$$

$$= \sum_{\substack{j \in \tilde{\rho}^{-1}(l) \cap \Psi_0 \setminus \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1} + \sum_{\substack{j \in \overline{\Upsilon}_{l,0} \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}) \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1}$$

$$= \sum_{\substack{j \in \overline{\Upsilon}_{l,0} \\ k \in [n_2]}} A_{j,k} w_k \mathbf{1}_j^{d_1} \ ,$$

where multiplying $\sum_{j \in \tilde{\rho}^{-1}(l) \cap \Psi_0 \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}), k \in [n_2]} A_{j,k} \mathbf{1}^{d_1 \times d_2}_{j,(2,j,k,\mathsf{lab})}$ and $\sum_{j \in \overline{\Upsilon}_{l,0} \cap \rho^{-1}(\mathsf{att})} w_k \overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')}$ yields $\sum_{j \in \overline{\Upsilon}_{l,0} \cap \rho_{\mathsf{lab}}^{-1}(\mathsf{lab}), k \in [n_2]} A_{j,k} w_k \mathbf{1}_j^{d_1}$, because, for each label lab and $j \in \overline{\Upsilon}_{l,0}$ with $\rho_{\mathsf{lab}}(j) = \mathsf{lab}$, there is exactly one attribute in $\mathcal{S}$ with the same authority l and label lab that matches it. Hence, for only that attribute, $\overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')}$ is included in the substitution vector for $\bar{r}_{l,\mathsf{att}}$.

Furthermore, the key polynomial $k_{3,l,(\mathsf{lab},\mathsf{att})}$ evaluates to the all-zero vector, i.e.,

$$\begin{aligned}
\bar{r}_{l,\mathsf{lab}}\left(\bar{b}_{l,\mathsf{lab},0}+x_{\mathsf{att}}\bar{b}_{l,\mathsf{lab},1}\right) \;:\; & \left(\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\cap\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}}\left(x_{\mathsf{att}}-x_{\rho(j)}\right)A_{j,k}\mathbf{1}^{d_1\times d_2}_{j,(2,j,k,\mathsf{lab})}\right.\\
& \left.+\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\setminus\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}}A_{j,k}\left(x_{\mathsf{att}}\mathbf{1}^{d_1\times d_2}_{j,(3,k,\mathsf{lab})}+\mathbf{1}^{d_1\times d_2}_{j,(5,k,\mathsf{lab})}\right)\right)\\
& \left(-\sum_{\substack{\mathsf{lab}'\in\chi_{l,\mathsf{att}}\\ k\in[n_2]}}w_k\left(\overline{\mathbf{1}}^{d_2}_{(3,k,\mathsf{lab}')}-x_{\mathsf{att}}\overline{\mathbf{1}}^{d_2}_{(5,k,\mathsf{lab}')}+\sum_{j\in\overline{\Upsilon}_{l,0}\cap\rho^{-1}(\mathsf{att})}\overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab}')}\right)\right)\\
= \; & -\sum_{\substack{j\in\overline{\Upsilon}_{l,0}\cap\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\cap\rho^{-1}(\mathsf{att})\\ k\in[n_2]}}\left(x_{\mathsf{att}}-\underbrace{x_{\rho(j)}}_{=x_{\mathsf{att}}}\right)A_{j,k}\mathbf{1}^{d_1}_{j}\\
& -\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\setminus\rho_{\mathsf{lab}}^{-1}(\mathsf{lab})\\ k\in[n_2]}}A_{j,k}w_k\left(x_{\mathsf{att}}\mathbf{1}^{d_1}_{j}-x_{\mathsf{att}}\mathbf{1}^{d_1}_{j}\right)\;=\;\mathbf{0}\;.
\end{aligned}$$

For the ciphertext polynomials with $j\in\Psi$, we have that $c_{1,j}=\mu_j+s_jb_{\tilde{\rho}(j)}$ is substituted by

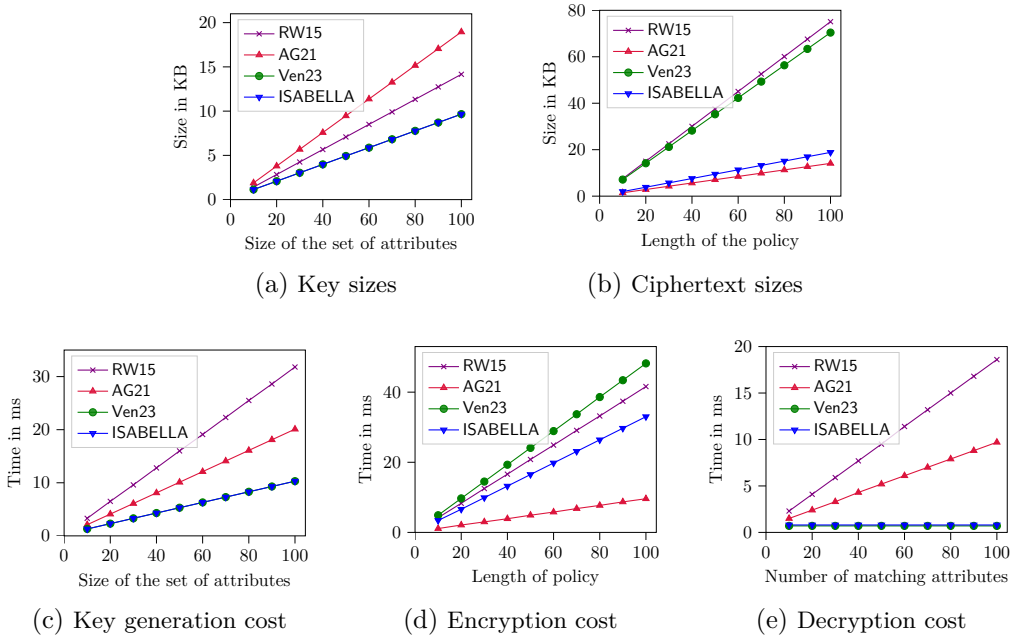
$$\begin{aligned}
&\sum_{k\in[2,n_2]}A_{j,k}\hat{\mathbf{v}}'_k\;-\;\mathbf{1}^{d_1}_{j}\left(\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\\ k\in[2,n_2]}}A_{j,k}\left(\mathbf{1}^{d_1\times d_2}_{j,(1,k)}+v_k\mathbf{1}^{d_1\times d_2}_{j,(1,1)}\right)\;-\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\\ k\in[n_2]}}A_{j,k}v_k\mathbf{1}^{d_1\times d_2}_{j,(4,1)}\right)\\
&\quad=\sum_{k\in[2,n_2]}A_{j,k}\left(\overline{\mathbf{1}}^{d_2}_{(1,k)}+v_k\overline{\mathbf{1}}^{d_2}_{(1,1)}\right)-\sum_{k\in[2,n_2]}A_{j,k}\left(\overline{\mathbf{1}}^{d_2}_{(1,k)}+v_k\overline{\mathbf{1}}^{d_2}_{(1,1)}\right)\;=\;\mathbf{0}\;,
\end{aligned}$$

and that $c_{2,j}=s_j(b_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),0}+x_{\rho(j)}b_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),1})$ is substituted by

$$\begin{aligned}
&\mathbf{1}^{d_1}_{j}\sum_{\substack{j'\in\tilde{\rho}^{-1}(\tilde{\rho}(j))\cap\Psi_1\cap\rho_{\mathsf{lab}}^{-1}(\rho_{\mathsf{lab}}(j))\\ k\in[n_2]}}x_{\rho(j')}A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(2,j',k,\mathsf{lab})}\;-\;\mathbf{1}^{d_1}_{j}\sum_{\substack{j'\in\tilde{\rho}^{-1}(\tilde{\rho}(j))\cap\Psi_1\setminus\rho_{\mathsf{lab}}^{-1}(\rho_{\mathsf{lab}}(j))\\ k\in[n_2]}}A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(3,k,\mathsf{lab})}\\
&\quad-\;x_{\rho(j)}\mathbf{1}^{d_1}_{j}\left(\sum_{\substack{j'\in\tilde{\rho}^{-1}(\tilde{\rho}(j))\cap\Psi_1\cap\rho_{\mathsf{lab}}^{-1}(\rho_{\mathsf{lab}}(j))\\ k\in[n_2]}}A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(2,j',k,\mathsf{lab})}\right)\\
&=\sum_{k\in[n_2]}x_{\rho(j)}A_{j,k}\overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab})}\;-\sum_{k\in[n_2]}x_{\rho(j)}A_{j,k}\overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab})}\;=\;\mathbf{0}\;.
\end{aligned}$$

For $j\in\overline{\Psi}$, we have that $c_{1,j}=\mu_j+s_j\bar{b}_{\tilde{\rho}(j)}$ is substituted by

$$\sum_{k\in[2,n_2]}A_{j,k}\hat{\mathbf{v}}'_k\;-\;\mathbf{1}^{d_1}_{j}\left(\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_0\\ k\in[2,n_2]}}A_{j,k}\left(\mathbf{1}^{d_1\times d_2}_{j,(1,k)}+v_k\mathbf{1}^{d_1\times d_2}_{j,(1,1)}\right)\;-\sum_{\substack{j\in\tilde{\rho}^{-1}(l)\cap\Psi_1\\ k\in[n_2]}}A_{j,k}v_k\mathbf{1}^{d_1\times d_2}_{j,(4,1)}\right)$$



(a) Key sizes (b) Ciphertext sizes

(c) Key generation cost (d) Encryption cost (e) Decryption cost

Figure 3: Comparison of decentralized CP-ABE schemes.

$$= \sum_{k\in[2,n_2]} A_{j,k}\left(\overline{\mathbf{1}}^{d_2}_{(1,k)} + v_k\overline{\mathbf{1}}^{d_2}_{(1,1)}\right) - \sum_{k\in[2,n_2]} A_{j,k}\left(\overline{\mathbf{1}}^{d_2}_{(1,k)} + v_k\overline{\mathbf{1}}^{d_2}_{(1,1)}\right) = \mathbf{0}\,,$$

and that $c_{2,j} = s_j(\bar{b}_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),0} + x_{\rho(j)}\bar{b}_{\tilde{\rho}(j),\rho_{\mathsf{lab}}(j),1})$ is substituted by

$$- \mathbf{1}^{d_1}_j \sum_{\substack{j'\in\tilde{\rho}^{-1}(\tilde{\rho}(j))\cap\Psi_0\cap\rho^{-1}_{\mathsf{lab}}(\rho_{\mathsf{lab}}(j))\\ k\in[n_2]}} -x_{\rho(j')}A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(2,j',k,\mathsf{lab})} - \mathbf{1}^{d_1}_j \sum_{\substack{j'\in\tilde{\rho}^{-1}(\tilde{\rho}(j))\cap\Psi_0\setminus\rho^{-1}_{\mathsf{lab}}(\rho_{\mathsf{lab}}(j))\\ k\in[n_2]}} A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(5,k,\mathsf{lab})}$$

$$- \mathbf{1}^{d_1}_j\rho(j) \sum_{\substack{j'\in\tilde{\rho}^{-1}(l)\cap\Psi_0\cap\rho^{-1}_{\mathsf{lab}}(\mathsf{lab})\\ k\in[n_2]}} A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(2,j',k,\mathsf{lab})} - \mathbf{1}^{d_1}_j\rho(j) \sum_{\substack{j'\in\tilde{\rho}^{-1}(l)\cap\Psi_0\setminus\rho^{-1}_{\mathsf{lab}}(\mathsf{lab})\\ k\in[n_2]}} A_{j',k}\mathbf{1}^{d_1\times d_2}_{j',(3,k,\mathsf{lab})}$$

$$= \sum_{k\in[n_2]} x_{\rho(j)}A_{j,k}\overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab})} - \sum_{k\in[n_2]} x_{\rho(j)}A_{j,k}\overline{\mathbf{1}}^{d_2}_{(2,j,k,\mathsf{lab})} = \mathbf{0}\,.$$

Lastly, $c'_j = \lambda_j + \alpha_{\tilde{\rho}(j)}s_j$ is substituted by

$$A_{j,1}\tilde{\mathbf{s}} + \sum_{k\in[2,n_2]} A_{j,k}\hat{\mathbf{v}}_k - \mathbf{1}^{d_1}_j \sum_{\substack{j'\in\tilde{\rho}^{-1}(\tilde{\rho}(j))\\ k\in[n_2]}} A_{j',k}v_k\mathbf{1}^{d_1}_{j'} = A_{j,1} + \sum_{k\in[2,n_2]} A_{j,k}v_k - \sum_{k\in[n_2]} A_{j,k}v_k = 0\,.$$

□

G Comparison of decentralized CP-ABE schemes

In order to demonstrate the effectiveness of our new compiler, we compare the cost of the decentralized CP-ABE scheme from [Ven23] (Appendix E.1) with our new compiler (cf. Fig. 2) with other constructions from the literature in Fig. 3. More specifically, we compare to RW15 [RW15] and AG21 [AG21], and the same decentralized scheme, but

obtained via the Ven23 compiler [Ven23]. Note that all these schemes are monotone, and therefore do not support negations.

Estimates for the computational efficiency based on benchmarks in RELIC. We estimate the computational costs of the schemes by obtaining benchmarks of various algorithms and extrapolating the results by analyzing the descriptions of the schemes. We analyze the efficiency in this way, because it allows us to compare the schemes more accurately and more fairly. Currently, the simplest and most popular way [RW13, AC17a, VAH23] to benchmark schemes is by using Charm [AGM+13]. However, Charm only supports curves that do not provide sufficient security anymore, and de la Piedra et al. [dlPVA22] show that benchmarking the schemes on these curves yields inaccurate and unfair comparisons. To compare the schemes more accurately and fairly, we estimate[2] the costs of the schemes by applying the ABE Squared approaches [dlPVA22]. In particular, we use their benchmarks in RELIC [AGM+], a cryptographic library for efficient implementations of pairing-based cryptography on state-of-the-art elliptic curves. This library has implementations for exponentiations, including fixed-base variants. In fixed-base exponentiation, the base g in g^x is fixed after setup, and as such, a precomputation table can be made to speed up the computation [BGMW92]. For all schemes, we also assume that the access policies are Boolean formulas, so that for decryption, it is ensured that $\varepsilon_j \in \{0, 1\}$ [LW11]. This optimizes decryption similarly for all schemes. Our theoretical estimations are for the BLS12-381 curve [BLS02, Bow], which provides approximately 128 bits of security [Gui20].

Evaluation. To compare the key and ciphertext sizes, we use the same approach as in [VA]. Since ISABELLA allows us to push elements in the ciphertext from the target group to the source groups, our ciphertext sizes are comparable to those of AG21 and more than 3 times smaller than those of RW15 and Ven23. The keys are generated as in Ven23 and are smaller than in RW15 and AG21.

When it comes to the runtime, our scheme gives the best performance for key generation and for decryption. Both are the same as in Ven23 and outperform the other schemes. Most notably, we require only a constant number of pairing operations in decryption. While AG21 is the fastest scheme for encryption, its decryption time grows with the number of matching attributes.

Jupyter notebook. The Jupyter notebook accompanying these theoretical performance estimates is available at `https://github.com/lincolncryptools/ISABELLA`.

[2] Although approximated theoretically, we expect our estimates to be close to the costs of actual implementations, as was shown in [VA23].

Contents

1 Introduction **1**
1.1 Technical overview of our contributions . . . 3
1.2 Roadmap . . . 7

2 Preliminaries **8**
2.1 Notation . . . 8
2.2 Access structures . . . 8
2.3 Pairings (or bilinear maps) . . . 8
2.4 Attribute-based encryption . . . 9
2.5 Specific predicate types . . . 10
2.6 Multi-authority ABE . . . 10

3 Our class of pair encoding schemes **12**
3.1 The special symbolic property . . . 13
3.2 Multi-key and multi-ciphertext security . . . 14
3.3 Security notions in matrix notation . . . 15
3.4 Handling corruption . . . 19
3.5 Equivalence with SSSP . . . 23
3.6 Computer-assisted proofs with ACABELLA . . . 24

4 Our generic compiler **26**
4.1 Distribution of encodings and FDHs . . . 26
4.2 Split-predicate mold for PES . . . 28
4.3 The ISABELLA compiler . . . 30

5 Security of our generic compiler **31**
5.1 Security model . . . 31
5.2 Static security under q-type assumptions . . . 33
5.3 Adaptive security in the GGM . . . 35

6 Applications and new schemes **36**
6.1 Multi-authority attribute-based encryption . . . 36
6.2 Functional adaptivity in ABE . . . 38

References **40**

A The ACABELLA extension **44**
A.1 Our parser and correctness checker . . . 44
A.2 Proof generation - finding a suitable kernel vector . . . 46
A.3 Proof generation - decomposing the kernel vector . . . 49
A.4 Verifying the proofs . . . 51

B Linear algebra tools **52**
B.1 Constructing the basis $\mathcal{W}_{\mathrm{cor}}$ . . . 52
B.2 Constructing the basis $\mathcal{W}_{\mathrm{cor,ki}}$ (when $|\mathbf{c}'| \geq 1$) . . . 52
B.3 Constructing the basis $\mathcal{W}_{\mathrm{cor,ki}}$ (when $|\mathbf{c}'| = 0$) . . . 55
B.4 Completeness of the algorithm . . . 56

C Proof of Theorem 3 **57**

D The generic group model and omitted proofs **63**
D.1 Generic hardness of (d_1, d_1', d_2)-spDBDH 63
D.2 Proof of Theorem 4 64

E Existing pair encoding schemes **65**
E.1 Decentralized large-universe ABE based on RW15 and AC17 65
E.2 ABE with attribute-wise key generation based on RW13 67

F New pair encoding schemes **68**
F.1 Decentralized large-universe KP-ABE from FDH 68
F.2 Completely unbounded decentralized CP-ABE with negations 72

G Comparison of decentralized CP-ABE schemes **78**